KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Science

Algorithm performance prediction for practical combinatorial optimisation problems

Tommy Messelis

Dissertation presented in partial fulfilment
of the requirements for the degree of
Doctor of Sciences: Informatics

June 2014

Algorithm performance prediction for practical combinatorial optimisation problems

Tommy MESSELIS

Supervisory Committee:
Prof. Dr. H. Deckmyn, chair
Prof. Dr. P. De Causmaecker, supervisor
Prof. Dr. ir. H. Blockeel
Prof. Dr. L. De Raedt
Prof. Dr. ir. G. Vanden Berghe
Prof. Dr. A. J. Parkes
(University of Nottingham, United Kingdom)

Dissertation presented in partial fulfilment of the requirements for the degree of
Doctor of Sciences: Informatics

June 2014

Uitgegeven in eigen beheer, Tommy Messelis, Etienne Sabbelaan 53, B-8500 Kortrijk (Belgium)

ISBN 978-90-8649-732-4

D/2014/10.705/40

Abstract

This dissertation investigates algorithm performance predictions in the context of combinatorial optimisation problems. The project is inspired by existing studies on running time predictions for complete search methods solving classical decision and optimisation problems. It considers more general performance criteria in the context of incomplete methods. One of the core concepts in this thesis is the notion of empirical hardness, denoting the apparent complexity of an instance as it is observed by a particular solver. We start by proposing a general strategy allowing for the construction of prediction models for the empirical hardness of practical problem instances. This strategy is formulated at a high level, allowing it to be applied in a broad variety of settings. We apply the strategy in two case studies, each focusing on a prototypical hard combinatorial optimisation problem with practical relevance. We consider the nurse rostering problem and a generalisation of the project scheduling problem. We investigate state-of-the-art metaheuristic algorithms for both problems and demonstrate the power of the methodology in this more complicated practical context. In particular, we present practical instantiations for all ingredients of the strategy. By extensive experimentation, we succeed in realising accurate performance prediction models. We furthermore build successful applications in the form of automatic algorithm selection tools, outperforming all of their state-of-the-art components individually.

Based on the experience gained through the doctoral project, we present a discussion on the application of the proposed strategy in practical settings. We focus on the important ingredients and discuss how they can be instantiated.

Beknopte samenvatting

Deze verhandeling onderzoekt hoe performantievoorspellingen mogelijk gemaakt kunnen worden binnen de context van combinatorische optimalisatieproblemen. Dit project laat zich inspireren door bestaand onderzoek naar het voorspellen van de benodigde rekentijd van complete oplossingsmethodes in de context van klassieke beslissings- en optimalisatieproblemen. Binnen dit onderzoek wordt de focus gelegd op andere performantiecriteria dan de benodigde rekentijd. Eén van de belangrijkste concepten in deze context is de notie van empirical hardness (of empirische moeilijkheid). Hieronder wordt de moeilijkheid van een probleeminstantie begrepen, zoals deze ervaren wordt door een specifieke oplossingsmethode. In deze verhandeling stellen we een strategie voor die toelaat modellen te construeren die deze moeilijkheid voorspellen. Deze strategie wordt op een bepaald abstractieniveau geformuleerd, zodat deze gebruikt kan worden in een brede waaier van toepassingsgebieden.

Binnen deze verhandeling wordt de voorgestede strategie toegepast in twee praktische casestudy's. Dit gebeurt voor het probleem van het opstellen van werkschema's voor verpleegkundigen en voor het probleem van het opstellen van uitvoeringsschema's voor grote projecten. Voor beide problemen wordt gekeken naar state-of-the-art oplossingsmethodes op basis van (meta-)heuristieken. Hierbij wordt aangetoond hoe de voorgestelde strategie op een succesvolle manier kan worden toegepast in dergelijke praktische context. Meer specifiek worden concrete invullingen gegeven voor de ingrediënten van de strategie die leiden tot nauwkeurige voorspellingsmodellen. Verder worden deze voorspellingsmodellen toegepast binnen het kader van automatische algorithmeselectietoepassingen. Hierbij wordt voor een gegeven probleeminstantie gepoogd te voorspellen welke oplossingsmethode (uit een gegeven verzameling) tot de beste resultaten zal leiden. Er wordt aangetoond dat een dergelijke toepassing van voorspellingsmodellen effectief tot een verbeterde algemene performantie ten opzichte van elk van de algoritmen individueel kan leiden.

Vanuit de ervaringen die opgedaan werden tijdens het uitvoeren van deze experimentele studie, wordt besproken hoe de voorgestelde strategie kan toegepast worden in een nieuwe praktische context. Hierbij wordt vooral aandacht geschonken aan de verschillende ingrediënten en aan de manier waarop deze concreet kunnen ingevuld worden.

Dankwoord

Less is More

– Ludwig Mies van der Rohe

Dit boekje, het resultaat na 8 jaar Kulak, een hindernissenparcours. De eindmeet, een nieuwe start ook. Op dit punt is het zwoegen voorbij, hoog tijd om enkele mensen te bedanken, mensen zonder wie dit resultaat onmogelijk was geweest.

Eerst en vooral wil ik mijn promotor, Patrick De Causmaecker, bedanken om mij welkom te heten in zijn startende onderzoeksgroep. Het was in het begin even zoeken naar de juiste richting, maar dankzij de steun en brede expertise van Patrick zijn we er uiteindelijk toch geraakt. Hij gaf me steeds voldoende vrijheid en vertrouwen, en waar nodig bood hij ook de juiste sturing. Niet alleen op het vlak van onderzoek, maar ook wat betreft onderwijs en het organiseren van (professionele én ontspannende) activiteiten.

Ik wil graag ook de leden van mijn begeleidingscommissie, Luc De Raedt en Hendrik Blockeel bedanken voor hun constructieve feedback op de tussentijdse presentaties. Zij waren de krachten aan de zijlijn die mij op tijd en stond vanuit een ander standpunt deden kijken naar mijn werk. Ook de andere leden van de jury, Greet Vanden Berghe, Andrew Parkes en voorzitter Hans Deckmyn wil ik bedanken voor hun kritische blik, zonder dewelke dit resultaat niet was geweest wat het nu is. Bijzondere dank gaat hierbij uit naar Greet, die vanaf het begin ook nauw betrokken was bij dit project.

Naast de promotor en de jury wil ik ook de Groep Wetenschap & Technologie Kulak bedanken voor het financieel ondersteunen van mijn traject. In het bijzonder gaat mijn dank uit naar Paul Igodt, die steeds met uiterste zorg en met kritisch oog gewaakt heeft over de goede loop van zaken. De wederzijdse appreciatie heeft het werken op de Kulak tot een zeer aangename ervaring gemaakt.

Naast de bazen zijn er uiteraard ook de Kortrijkse onderzoekscollega's die tijdens en tussen de pauzes voor de leuke sfeer zorgen. Paco, Stefaan en Nguyen op het bureau; Stefan, Mark, Ruben, Igor, Kuchi, Ahmet, San, Bidzina en de ITEC collega's daarbuiten. In het bijzonder wil ik Stefan bedanken. Het klikte meteen in Venetië en een traditie was geboren. Conferenties zullen nooit meer hetzelfde zijn zonder jou.

Collega's en vrienden op de Kulak lopen doorheen alle disciplines heen. Wat de meesten van hen verbindt is de koffiekamer. Dank aan de collega's die het er dagelijks om 10u en 16u aangenaam vertoeven maken.

Verder wil ik ook de badmintonvrienden en de kaartersclub bedanken. Het deed deugd even alles van me af te kunnen zetten en voluit pluimpjes te slaan, of onnozele pico's te spelen. Merci Mark, Jonas, Gert-Jan, Thijs, Frederik, Wim, Karel, Steven, Pieter, Brecht, Brecht[2], Piet en Joepie.

Niet in het minst wil ik ook mijn vrienden bedanken voor het altijd aangename gezelschap, de lekkere etentjes en de ontspannende speelavonden. Bedankt Evelien en Sandy, Maarten, Leander; Alexander en Ellen, Delfien; Maarten en Sarah, Wieland en Tini, Thijs en Nelle; Thomas en Joris; Stijn en Annelies, Stefan en Barbara; Mark en Lien, Igor en Gwen; Yves en Ann, Julien en Dageraad.

Verder gaat mijn oprechte dank uit naar mijn ouders, die mij alle kansen geboden hebben en mij altijd onvoorwaardelijk gesteund hebben. Bedankt ook aan mijn broer en zussen, die altijd voor leven in de brouwerij zorgen.

Last but not least; Jonathan, zonder jou was dit alles niet mogelijk geweest. Zonder jouw steun, je oneindige geduld, je concrete aanmoedigingen, opnieuw en opnieuw, zonder jouw aanstekelijke lach en je lekkere eten, zonder jou had ik niet tot dit resultaat kunnen komen.

Bedankt!

Tommy
Kortrijk, Juni 2014

Less is a bore

– Robert Venturi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This first chapter sets the scope of this dissertation. It is our aim to introduce the reader to the main topic of this doctoral project: *building algorithm performance prediction models for practical combinatorial optimisation problems*. We begin with a brief introduction into the field of combinatorial optimisation. We present a number of challenges in this context, exposing the main research questions and addressing their practical relevance. We furthermore discuss the structure of the text, which roughly follows the way the experimental study was carried out. We conclude this introductory chapter with a short overview of the main contributions.

1.1 Context

In everyday life, people are frequently faced with practical problems. They may have to travel to their offices through a possibly congested network of streets and obstacles, requiring them to choose a transportation mode and a specific trajectory. They have to decide what to eat, and based on the content of their refrigerator, they may have to go out and shop for the necessary ingredients. Several factors need to be weighed in, and unforeseen circumstances may require a change of plans. There is a need for flexibility; both for the people themselves, and for the systems supporting them in making all sorts of decisions. Some of these decisions are simple, but more complex challenges can be encountered too. Absences at work may influence the workload of colleagues, or even require interim staff to be hired. A common property is that a number of choices are to be made. It is hereby not always clear what the best options are. In

many practical settings where quick actions or reactions are required, the final outcome of these choices is not the best possible scenario. Optimality is in many cases a very difficult, if not impossible, goal to achieve. Fortunately however, such optimality is not always strictly required. In many cases, people are satisfied with fairly good solutions.

This situation is not very different for researchers and practitioners solving operational problems. Their problems tend to be even more difficult than the everyday challenges. The field of *Combinatorial Optimisation* is concerned with problems where the goal is to find the best option out of a finite set of possible solutions. This set of possible solutions may be very large, even for relatively small problem sizes. Typically, the number of possible solutions grows exponentially with the size of the problems. Searching for the best solution can take a very long time, even for small problem instances.

There are two types of combinatorial search methods: complete and incomplete methods. The difference lies within the guarantees that are given regarding the produced solutions. Complete search methods deliver an optimal solution, while incomplete search methods can not give such guarantees.

Basic *brute-force search* methods (also called *naive search* or *exhaustive search* methods) are complete algorithms. They enumerate all possible solutions and simply output the best encountered solution. Sometimes, clever structures can be employed allowing significant reductions of the search space. This is the case for e.g. backtracking and branch-and-bound algorithms. Although significant speed-ups can be achieved, such complete search methods may still require long calculation times before the optimality of their solutions can be guaranteed. This turns them infeasible for many practical combinatorial optimisation problems of realistic size.

When the search space is too large, there is no other option than to apply incomplete search methods. An important class of such methods are *heuristic search procedures*. *Heuristics* can be thought of as simple rules of thumb. They describe basic strategies which intuitively lead to fairly good solutions. Heuristic search procedures do not aim at exploring the complete search space. Instead, they can be used to generate solutions or to transform solutions into better solutions, leading to a final solution for which no formal guarantees can be made. Such methods explore only (small) parts of the search space, making them much faster than complete search methods, which is their most important characteristic.

Heuristics are the basic building blocks for higher-level search strategies such as *metaheuristics*. There is no commonly agreed definition of the term *metaheuristic*. In this dissertation, we consider metaheuristics to be higher-level

strategies to guide, build or select lower-level heuristics or search processes in order to find good enough solutions in a reasonable amount of time. The term *good enough*, as e.g. defined in terms of an objective function, denotes solutions that are better than (or as good as) the solutions which are currently possible to achieve. The current state of the art can be the result of existing heuristic or metaheuristic procedures. In many practical settings however, professionals are still solving problems by hand. Metaheuristics are considered to be valuable as soon as they can produce similar or better results than the current state of the art, especially in the case where complete methods are infeasible. Metaheuristics sacrifice a certain property (e.g. completeness, accuracy, or optimality) in return for significantly reduced execution times. It is only rarely the case that strict bounds on the quality of the solutions with respect to optimal ones can be delivered. Consequently, many metaheuristics require an external stopping criterion. When ultimately this criterion is met, they report the best found solution so far. The meaning of a *reasonable* amount of time can thus be regulated by the stopping criterion that is being applied and varies from application to application.

An important class of metaheuristics are *local search* methods. Local search algorithms view the solution space as a graph. The (partial) solutions are represented by vertices and edges indicate that two solutions are *neighbours*. A neighbourhood defines a set of operations (or heuristics) transforming one solution into another. Two solutions are called neighbours when one can get from one solution to the other by applying one or more such operations. Local search methods start from an initial solution and transverse this graph searching for the best possible solution. Steepest-descent, best-first, or even stochastic criteria can be used to select the next vertex in the search process. This process typically leads to a local optimum. Several techniques exist to escape from such local optima, allowing the search process to continue in other regions of the search space. A simple (but effective) example is to just randomly start from a new initial solution. Another possibility is to temporarily allow worsening moves in order to reach other basins of attraction.

Another class of metaheuristics are population-based techniques. Evolutionary algorithms, for example, mimic the evolution of populations based on the principle of 'survival of the fittest'. Solutions are represented by individuals and are assigned a fitness value. During a set of evolutions, the fittest individuals are combined in the hope for producing even better solutions. Another example of a population-based technique is ant colony optimisation. In this paradigm, a population of ants explores a search space and leaves pheromone trails signalling promising regions, which are then more intensively searched. When a trail gets cold, the search diverges and other regions are to be explored.

Metaheuristics are thus defined as higher-level procedures operating on lower-level heuristics. In a sense, metaheuristics are thought to be problem-independent. Indeed, they express a way of exploring a search space in order to find good solutions. However, this idea of problem-independence is only valid at this higher level. The lower-level heuristic on which metaheuristics operate are not problem-independent. When a researcher or practitioner wants to build a metaheuristic algorithm for a certain problem, a number of specific decisions have to made. The way solutions should be represented, which elementary transformations are possible, how genetic cross-over can be represented, etc. are only a few examples. All these decisions are non-trivial and lead to a specific instantiation of a metaheuristic, which is only limitedly applicable. Hence, the concept of a metaheuristic is problem-independent, the actual algorithm able to solve a problem is not. These concepts, however, are used interchangeably throughout the literature. In this dissertation, we will mainly use the term 'metaheuristic' to denote a specific algorithm instantiation able to solve a particular problem. At some points, we will also use the term to specify the higher-level idea lying behind such an instantiation.

As an example, *tabu search* is a local search metaheuristic in the sense that it is an idea, a strategy allowing the guidance of a set of lower-level heuristics or operations, guiding a search process. It specifies how a certain *neighbourhood* is to be explored.

The idea is to select a certain *move* allowing to go from one solution to another, and to use memory to avoid going back to recently visited solutions. Certain properties of the move itself can also be remembered in order to avoid selecting the 'same' move again during a consecutive number of steps. However, for this metaheuristic to be applicable to a specific problem, e.g. a travelling salesman problem, a number of decisions have to be made: one must decide on a representation for the (intermediate) solutions; the neighbourhoods must be defined, i.e. a number of moves must be defined allowing to get from one solution to another; one must decide on the representation of the specific properties of moves that should be avoided; the length of the tabu list should be set; etc. All these decisions are non-trivial and lead to a specific instantiation of a tabu search metaheuristic. In this dissertation, we will use the term metaheuristic to denote both the higher-level idea and its specific instantiation interchangeably.

Using metaheuristics instead of complete search approaches greatly reduces the time necessary for finding good solutions. However, deciding on an appropriate stopping criterion is not always straightforward. In the context of this dissertation, we are interested in a number of situations where such speed-ups are still not sufficient. We will investigate how predictive systems can play a supportive role in such situations, which is the core theme of this doctoral project.

1.2 Challenges

Like all algorithms, metaheuristics have their own particular strengths and weaknesses. Some algorithms work well on some instances, while their results on other instances might be much worse. This statement is supported by what is observed in many practical settings. It appears that there does not exist a single best algorithm outperforming all others on all instances of a specific problem. Instead, the best algorithm depends on the particular instance that needs to be solved.

Immediately, an interesting setting for a predictive system emerges. When a set of competitive algorithms is available, a predictive system can support the choice of which algorithm to run on a given instance simply by comparing performance predictions. When computing resources are scarce, a predictive system may lead to a more efficient use of resources. This is the case when many problem instances need to be solved consecutively. Running all algorithms on all instances would be infeasible. A predictive system selecting the best method for a given instance could greatly reduce the overall running time, while at the same time significantly improve on the average performance of using only one algorithm.

In other situations, it is sometimes useful to have an immediate idea of the quality of a solution resulting from a given algorithm, without the need to actually calculate this solution. An example can be found in settings where many different alternatives need to be evaluated. A negotiation system for the exchange of resources, e.g., lets a number of parties decide on how to work together. Each party must therefore quickly evaluate how well it can function in different scenarios. Predictive systems allow for such quick estimates, without the need to run time-consuming algorithms.

Having predictive systems for algorithm performance furthermore allows for a deeper investigation of why certain algorithms work well on certain instances. Predictive models can provide insight into the properties making a problem easy or hard for a specific algorithm. Such insights are important in themselves. They may furthermore lead to the design of better algorithms, focusing on those characteristics that matter.

1.3 Research questions

In this dissertation, we focus on predictive systems for algorithm performance in the context of practical combinatorial optimisation problems. This project is

positioned at the intersection of the fields of artificial intelligence and operational research. The extensive experimental study is based on an existing framework allowing for the prediction of running times of complete search algorithms solving formally stated problems (Leyton-Brown et al., 2006). This framework is not directly applicable in a practical situation where metaheuristics are employed to solve hard combinatorial optimisation problems. It was designed for approaches that are guaranteed to either find a solution, or to stop and prove that there is no solution. For such decision problems, the outcome of an algorithm is binary. Either the algorithm produces a solution (and every solution is equally good), or it proves that there is no solution. In the case of optimisation problems, there is a difference in quality between solutions. There are numerous possible solutions and the aim is to find the best one, given some definition of an objective function. Furthermore, when metaheuristics (or other incomplete search methods) are applied, the outcome is not necessarily optimal. Moreover, running time is often considered as a predetermined stopping criterion. Consequently, the framework will have to be adapted to be applicable in our context. This leads to the main research question addressed in this dissertation:

> *How can this framework for running time predictions of complete search methods be adapted to handle more practical settings where optimisation problems are solved using metaheuristic methods?*

We answer this question by proposing a number of adaptations to the framework such that other performance criteria for other types of algorithms could be considered as well. As with the concept of metaheuristics, this framework is rather general and formulated at a high level. The answer to this question is thus also a theoretical, general framework, constructed around a set of ingredients. For it to be applied in a practical setting, these ingredients need to be instantiated. This main research question (and its answer) thus unavoidably leads to more detailed questions regarding the actual settings in which performance predictions are desired.

We will therefore investigate such questions in two different settings: (1) for nurse rostering problems and (2) for project scheduling problems. The main body of this dissertation is thus concerned with answering the following more detailed questions:

> *How can the framework be applied to the nurse rostering problem?*

and

> *How can the framework be applied to the multi-mode resource-constrained project scheduling problem?*

Answering these questions leads to a number of important instantiations of the main ingredients of the framework, upon which practical applications can be built.

The experience gained from investigating these case studies allows us to address a more general research question:

> *How should the ingredients of the framework be instantiated in any practical setting?*

This question does not have a straightforward answer. There is no concise set of rules or guidelines applicable to all possible practical settings. We will present a thorough discussion on how the ingredients were instantiated in the two presented case studies. Through this discussion, we will formulate a number of good practices, interweaved with examples from our experience. The aim is to bridge the gap between the generality of the proposed framework and its practical application, to facilitate other researchers and practitioners in building such applications.

1.4 Structure of the text

The body of this dissertation is structured along the way the experimental study was carried out. In more detail, the subsequent chapters focus on the following subjects:

In **Chapter 2**, we position our work in the current state of the art. We build a relevant context through a combination of a literature overview and the introduction of novel ideas. We start by discussing the concept of *empirical hardness*, which is a key concept being used throughout the entire text. We then present an existing framework for building running time prediction models and discuss the modifications needed for its application in the context of this doctoral project. An important application can be found in automatic algorithm selection tools. In this context, we introduce a number of quantitative concepts supporting researchers in deciding which algorithms to include in such tools. We furthermore briefly discuss the related fields of algorithm configuration, hyper-heuristics and algorithm footprints, positioning our work in the broader literature.

The focus of **Chapter 3** is the case study on nurse rostering problems. The chapter starts with a short literature overview and presents a mathematical model defining the specific nurse rostering problem under study. This chapter

consists of two main parts: a proof-of-concept study and a practical case study. In the proof-of-concept study, we investigate whether the generalised framework can indeed lead to accurate algorithm performance predictions for metaheuristics solving combinatorial optimisation problems. The considered problems are deliberately kept small and a comparison is made between an incomplete and a complete search algorithm. The good results lead to the application of these ideas to a larger and more realistic setting based on a scientific and international nurse rostering competition. A practical application in the form of an automatic algorithm selection tool consisting of state-of-the-art algorithms is built. This tool effectively outperforms any of its components individually. The main results of this chapter have been published as:

- Messelis T., De Causmaecker P. An algorithm selection approach for nurse rostering. *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011), Gent, Belgium, 3–4 November 2011*, pages 160–166, 2011.
- Messelis T., De Causmaecker P., Vanden Berghe G. Algorithm performance prediction for nurse rostering. *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2013), Gent, Belgium, 27–28 August 2013*, pages 21–38, 2013.

In **Chapter 4**, we present the second case study of our doctoral research project. The chapter focuses on a generalisation of the commonly used resource-constrained project scheduling problem. A mathematical description of the problem is presented and a brief discussion of the relevant literature is included. We go into more detail on a number of complexity indicators proposed in the literature. The construction of algorithm performance prediction models is investigated for two state-of-the-art metaheuristics. This leads to an interesting setting for applying of such models in an automatic algorithm selection tool. Large parts of this chapter have recently been published as:

- Messelis T., De Causmaecker P. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511-528, 2014.

Chapter 5 takes a step backwards and bundles the experience gained from the previous chapters. We discuss *how* the important ingredients of the proposed procedure can be instantiated in a specific setting. This discussion is interweaved with examples from the case studies in Chapters 3 and 4.

In **Chapter 6** finally, we reformulate the research questions and discuss how this dissertation has answered them, focusing on the main contributions of this

doctoral project. We conclude with a brief discussion of some opportunities for further research.

At the end of this dissertation, we include four appendices. Two of these include detailed descriptions of the developed feature sets discussed in Chapters 3 and 4. The other two contain detailed descriptions of the selected feature sets for several prediction models discussed in this dissertation.

1.5 Contributions

In this introductory chapter, we have set the scope for the research presented in the remainder of this text. We introduced the reader into the field of combinatorial optimisation and sketched the outline of the doctoral project investigating the construction of algorithm performance prediction models in this context. In summary, we present here the main contributions of the work you are about to read:

- Based on an existing framework for building running time prediction models, a **generic framework for building algorithm performance prediction models** is proposed. This framework is applicable in the context of metaheuristic algorithms solving hard combinatorial optimisation problems.
- We present a **theoretical framework** allowing the **characterisation of a set of algorithms** in terms of their reciprocal **competitiveness** and the combined **potential impact** on an instance set, supporting the decision of which algorithms to use in an algorithm selection tool.
- A **case study on nurse rostering problems** is presented. This case study leads to the following contributions:
  - a set of **accurate performance prediction models** for state-of-the-art metaheuristic algorithms
  - a **feature set for nurse rostering** allowing the characterisation of a problem instance as a vector of 305 feature values
  - a practical application of performance prediction models in the form of an **automatic algorithm selection tool**, demonstrating that this approach leads to significant performance improvements over each of the components individually

- A **case study on multi-mode resource-constrained project scheduling problems** is presented. This case study leads to the following contributions:
  - a set of **accurate performance prediction models** for state-of-the-art metaheuristic algorithms
  - a **feature set for project scheduling** allowing the characterisation of a problem instance as a vector of 686 feature values , easily extendible to include other problem variants
  - a practical application of performance prediction models in the form of an **automatic algorithm selection tool**
- We present **a thorough discussion on *how* the proposed procedure can be instantiated in a practical setting**, bridging the gap between the general framework and its implementation in a specific context.

Chapter 2

Algorithm performance prediction

The aim of this chapter is to position our work in the current state of the art. We will realise this goal by building a relevant context for this dissertation. This includes reviewing key publications from the relevant fields, revealing a number of opportunities for novel research. These opportunities lead to the introduction of a number of new ideas, which will form the basis for the work presented in the following chapters.

This chapter thus includes both a literature overview, and the introduction of novel ideas, presented in an intertwined manner. Through this approach, the chapter follows a natural order for introducing and discussing the relevant context.

In Section 2.1, the notion of empirical hardness is introduced. This is one of the core concepts used throughout the entire text. Section 2.2 focuses on building prediction models for the empirical hardness of computational problems. An existing framework allowing for the construction of running time prediction models will be discussed. Furthermore, we will propose a generalisation of this framework allowing for a broader applicability. We will go into more detail on two important elements of such prediction models. Section 2.2.1 will discuss the features that lie at the basis of such empirical hardness models. We will review a number of publications on this topic and will present a prototypical example of such a feature set for a well-studied problem domain. Section 2.2.2 will discuss the construction of such models through the application of machine learning techniques. It is not our aim to go into detail on the different techniques. Instead,

we want to briefly discuss a number of practical considerations regarding the application of such techniques. A straightforward application of performance prediction models can be found in algorithm selection tools. Section 2.3 is dedicated to this topic. It furthermore includes the introduction of a novel theoretical framework supporting the choice of algorithms to consider in such applications. Sections 2.4−2.6 will briefly discuss the related fields of algorithm configuration, hyper-heuristics and algorithm footprints. Finally, in Section 2.7, we summarise the main contribution of this chapter, emphasising the novel ideas.

Please note that it is not the aim of this chapter to cover all publications in the relevant domains. Instead, we focus only the main publications, building a context for this doctoral project.

2.1 Empirical hardness

One of the most important concepts used throughout this dissertation is the notion of *empirical hardness*. We define this concept as follows:

> The *empirical hardness* of a problem instance is the apparent complexity of an instance, as it is experienced by a particular solver. This complexity is measured in terms of a performance criterion w.r.t. the considered solver.

Please note that the idea underlying this concept is not new. Several authors have been interested in the individual hardness of instances when solved with a particular algorithm (See e.g., Selman et al., 1996; Kostuch and Socha, 2004; Nudelman et al., 2004; Leyton-Brown et al., 2006). Leyton-Brown et al. (2006) introduced the term *empirical hardness* to denote the running time of an algorithm solving a particular instance. Kostuch and Socha (2004) did not introduce new terminology, but simply used the term *hardness* to denote the solution quality that can be obtained by an algorithm. In our definition (which is thus not necessarily original), we allow the empirical hardness to be measured by an arbitrary performance criterion.

The terms *'empirical'* and *'hardness'* both refer to an important aspect of this concept. The term *'empirical'* refers to the fact that this complexity is inextricably linked to the application of a particular solver. As already briefly mentioned in Section 1.2, different algorithms may perform very differently on the same problem instance. For one algorithm, a certain instance may be considered hard, while another algorithm might easily find a good or optimal

solution (and thus consider the instance easy). Putting a complexity label on an instance is highly dependent of the algorithm used to solve it. In order to have an idea of this complexity, an algorithm must be run, hence the term *empirical*. Furthermore, the apparent *hardness* or complexity must be quantified. This may be done in several ways, but is always related to a performance criterion of the considered algorithm. It could e.g. be measured as the time an algorithm needs to solve the instance. Another option is to look at a certain qualitative property of the resulting solution.

The empirical hardness of an instance is a result of two factors: (1) the intrinsic complexity and (2) the particularities of the algorithm being used to solve it.

The *intrinsic complexity* could be defined as follows:

> The *intrinsic complexity* of a problem is the result of a combination of problem-specific properties making it hard or easy to solve, regardless of the algorithm being used.

This complexity is a characteristic inherently linked to the problem. Its definition here is intentionally rather vague, we merely want to denominate those characteristics enclosed in an instance influencing its hardness, regardless of the algorithm being used. An example may clarify this. The problem of deciding whether there is an intersection between two lines in a plane is easier than that of deciding whether this is the case in a three-dimensional space. It is easy to understand that this is the case, regardless of how you try to solve this problem. The extra dimension makes the problem as such more difficult. The intrinsic complexity could be measured in terms of classical complexity theory, classifying problems as being e.g. P, NP or NP-hard.

The second factor influencing the empirical hardness is the algorithm being used to solve instances. One instance can be considered hard by one algorithm, yet easy by another. Its intrinsic complexity is the same and the difference in empirical hardness is in this case completely due to the way both algorithms solve the instance. In general, instances differing in empirical hardness may differ in intrinsic complexity too. Intrinsic hardness may not discriminate between two NP-hard problems while empirical hardness might be able to give an indication as to which algorithm is better suited for solving them (See e.g. Woeginger, 2003).

There have been a number of studies on the empirical hardness of individual instances or certain distributions of NP-complete or NP-hard problems. Most of this work concerns decision problems (problems where the answer is either `yes` or `no`). One historical result of such a study is the discovery that random 3-SAT

problems[1] are the hardest when the clauses-to-variables ratio is around 4.26 (Selman et al., 1996). This result is valid for solvers based on the Davis-Putnam-Logemann-Loveland (DPLL) method of Davis et al. (1962), solving instances with many variables. When there are fewer variables, the critical ratio is slightly higher. In their experimental study, Selman et al. gradually varied this ratio while generating random instances. They discovered a clear easy-hard-easy pattern, showing that the running time of their solver was significantly higher for the instances in the region around the ratio of 4.26. Furthermore, the authors demonstrated that this region corresponds to a phase transition in the probability that the random formula is satisfiable; i.e. a property not depending on the algorithm.

When looking at optimisation problems, things are far more complicated. It is not always possible to look at phase transitions since the solutions to such problems are not in the form of binary answers. The objective is to find an optimal solution with respect to some qualitative measure. This measure is often represented by an objective function. In this case, the task is to find a solution with an as high as possible quality value. However, such problems may be transformed into decision problems. One can ask whether there exists a solution for which the value of the objective function is below or above some threshold. Doing so, one can investigate phase transitions by gradually varying certain properties or thresholds. For many computational problems however, it is not at all clear which properties to vary in order to find such phase transitions. This is certainly the case when there are many different characteristics that *can* be varied. This is also the case for problems for which many, highly parametrised instance distributions exist.

2.2 Performance prediction

Performance prediction, or in other words: empirical hardness prediction, has first been introduced by (Leyton-Brown et al., 2006).[2] They presented an eight-step procedure allowing for the construction of running time prediction models. Their methodology is at the basis of this dissertation and consists of the following steps:

[1] Random 3-SAT problems are a subclass of general propositional satisfiability (SAT) problems. The aim in SAT problems is to find a model (i.e. a boolean assignment to the variables) that renders a given logical formula true. In 3-SAT problems, this formula is expressed in conjunctive normal form (CNF) and each clause consists of exactly three variables.

[2] This paper was actually a contribution to the Eighth International Conference on Principles and Practice of Constraint Programming in 2002 (CP2002). Publication of the proceedings took until 2006, which explains why certain later work appeared earlier than 2006.

1. Select an optimisation algorithm.
2. Select a set of problem instance distributions. For each parameter, decide on the range of the possible values.
3. Choose and fix the problem size.
4. Select a set of polynomial-time computable, distribution-independent features characterising problem instances.
5. Generate data: sample the instance distributions using parameter values in the decided ranges, until a sufficiently large dataset is generated.
6. Determine the running time and feature values for each instance in the dataset.
7. Eliminate redundant or uninformative features.
8. Learn a function of the features predicting the running time of the algorithm.

The application of this framework leads to a prediction model for the running time of the chosen algorithm, when applied to instances in the considered distribution. This model is based on specific instance properties or features.

Leyton-Brown et al. (2006) applied this strategy to the winner determination problem in combinatorial auctions. They considered a complete algorithm and developed a set of 35 instance features, which they thought could be relevant for the empirical hardness of the instances. Linear regression techniques were applied for building prediction models for the logarithm of the running time. Please note that fixing the problem size in Step 3 results from the well-known fact that auctions with a larger number of goods and bids resulted in longer running times. The aim was to focus on other, unknown sources of empirical hardness.

In a rather straightforward elaboration, Almajano et al. (2010) applied this methodology to the winner determination problem in mixed multi-unit combinatorial auctions. Again, linear regression techniques were used for the construction of prediction models, resulting in accurate performance predictions.

Nudelman et al. (2004) applied this strategy to investigate the empirical hardness (defined in terms of running times) of SAT problems. They demonstrated that accurate performance prediction models can be built for two classes of SAT problems, based on an extensive feature set for such problems. Furthermore, they developed a practical application of such models in an automatic algorithm selection tool, called SATzilla. This portfolio solver has since become an

important contestant and winner of several SAT-competitions. At this point, we will not go into further detail on algorithm selection. We will however extensively discuss this topic in Section 2.3.

Kostuch and Socha (2004) investigated similar ideas for the university course timetabling problem. The authors do not refer to any work of Leyton-Brown et al. (2006) or Nudelman et al. (2004), nor do they use the term *empirical hardness*. Their approach is however very similar and also employs linear regression techniques for building performance prediction models. Their aim is to predict the quality of the solutions obtained by a metaheuristic. Their results however, are limited, achieving an average prediction error of 17%, which is rather high.

Very recently, Hutter et al. (2014) presented an extensive overview of state-of-the-art methods for algorithm runtime predictions in the context of complete propositional satisfiability solvers, mixed integer program solvers and travelling salesman algorithms. They focus on improving prediction accuracy for highly parametrised algorithms and introduce new features for the considered problem domains. They conclude that random forests and Gaussian processes offer the best prediction accuracy in their settings. By incorporating the algorithm parameters as inputs for the learning process, they allow for the application of such predictors in automatic algorithm configuration tools. We discuss this related topic in Section 2.4.

In this dissertation, we will apply similar ideas for building empirical hardness models for incomplete search methods for practical combinatorial optimisation problems. Our approach is inspired by the work of Leyton-Brown et al. (2006). The application of such ideas in our context requires a number of adaptations to the framework. We therefore generalise and simplify the methodology and propose the following five-step procedure:

1. *Instance distribution:*
   Select an instance distribution, i.e. decide on the properties of the instances that will be considered.

2. *Algorithm set:*
   Select a number of algorithms and decide on the related performance criteria quantifying the empirical hardness of the instances.

3. *Feature selection:*
   Select or create a feature set characterising the instances in the distribution of Step 1.

4. *Data generation:*
   Construct a training set, calculate all feature values, and determine the performance of all algorithms selected in Step 2.

5. *Performance prediction models:*
   Experiment with different machine learning techniques in order to build accurate performance prediction models.

This generalised procedure has a broader field of potential applications than the original formulation. The most important adaptation follows from our definition of the concept of *empirical hardness*. Instead of focusing on running times, the methodology should support arbitrary performance criteria. In Step 2, we therefore include the selection of a performance criterion as a measure for the empirical hardness of the considered instances. As a consequence, we also generalise Step 5 such that other machine learning techniques than numeric function learning can be applied. Sometimes, it might be interesting to measure the empirical hardness of instances as a categorical value (e.g. easy/moderate/hard). In such cases, classification algorithms can be used to build empirical hardness models. Another difference is that our approach allows the construction of empirical hardness models for a set of algorithms, while the procedure of Leyton-Brown et al. (2006) considers only one algorithm. Evidently, their procedure can be repeated when predictions for more than one algorithm are required.

It is important to note that the specific instantiation of the ingredients of this framework is crucially important for its success in any context:

- In the first step, a problem instance distribution is to be chosen. This distribution has a large impact on the applicability of the results. It sets, and thereby limits, the scope of the empirical hardness models and hence, the resulting prediction models are only expected to be valid within this scope. It is thus important that the problem instance distribution resembles the instances for which performance predictions are to be made in the end. In a real-world setting for example, such a distribution could be derived from observing parameter ranges of the available real-world data. The learned predictors are inevitably related to this distribution and can be expected to perform well as long as the distribution does not change.

- In Step 3, a feature set is to be selected characterising the instances in the distribution. It is important that (at least some of) these features effectively relate to the empirical hardness of the instances. Building such a set is (as we will see) not straightforward. We dedicate Section 2.2.1 to this topic.

- In the last step of the methodology, the actual performance prediction models (or empirical hardness models) are to be constructed. The methodology allows experimentation with a range of machine learning techniques. It is important to properly prepare the data before applying such techniques, in order to avoid problems like over-fitting. Moreover, not all techniques will lead to accurate prediction models. A discussion on data preparation and how to evaluate and compare machine learning techniques is presented in Section 2.2.2.

2.2.1 Constructing problem instance feature sets

It has been noted that finding a good set of features is not straightforward (see e.g. Rice, 1976; Smith-Miles, 2009; Smith-Miles and Lopes, 2012). At the same time, it has been observed that the success of empirical hardness models is highly dependent on the quality of the feature sets (see e.g. Rice, 1976; Ramakrishnan et al., 2002; Xu et al., 2008). Building a qualitative feature set is of utmost importance for the accuracy of algorithm performance predictions.

Smith-Miles and Lopes (2012) review literature on the selection of a suitable feature set for a number of combinatorial optimisation problems. They have two distinct objectives in mind. They want to understand *how* good features can be found. And, they want to explore the question as to *why* these features work well. They want to investigate why certain algorithms work well for specific types of problem instances, and how these relationships can be described in terms of feature values. Regarding their first goal, the authors discover some similarities between features developed for different problems. After all, certain concepts are recurring in different combinatorial optimisation problems and this reflects in the instance features identified for these problems. Nevertheless, they conclude that in general, there is little overlap of ideas between different problem domains. In our view, this lack of commonality is an opportunity. In Chapter 5, based on our experience gained throughout this doctoral project, we will discuss in detail how such feature sets can be constructed. Through numerous examples, we aim at helping other researchers and practitioners in building qualitative feature sets in their own context. On the second goal, i.e. explaining why certain methods work well on certain instances and other methods do not, the results of Smith-Miles and Lopes (2012) are far less clear. They can not find clear or simple explanations to why one algorithm works better than another on certain problem instances. They conclude that there is still much more work to be done and refer to *algorithm footprints* as a step in the right direction. We refer to Section 2.6 for a discussion on such algorithm footprints.

Features can generally be divided into two categories: domain-specific features and domain-independent features. Domain-specific features are based on the properties of the problem description and are therefore linked to a particular problem domain. Domain-independent features can be applied to any problem domain and are hence more general in nature. We will review both types separately in the following subsections and will finish with an extensive example on propositional satisfiability problems.

Domain-specific features

Throughout the years, there have been many studies using domain-specific features for a variety of problem domains. It is not the aim of this section to review all literature on this topic, we will rather summarise some common ideas and refer to Smith-Miles and Lopes (2012) for an overview of domain-specific features for a number of combinatorial optimisation problems: assignment problems, travelling salesman problems, knapsack problems, bin-packing problems, graph problems, and timetabling problems.

An important aspect of a problem instance is its size. Size can be interpreted in several ways, depending on the problem description. It is therefore common to consider different measures for the size of the instances as features.

Additionally, depending on the types of constraints, or the concepts present in the considered problem, a number of features can be found relating to the constraint values and matrix or graph notations.

- For problems with capacity constraints (like e.g. knapsack problems and bin-packing), it is common to introduce features concerning the available slack and to use ratios of the requirements to the capacities. The ranges of the requirements are also important features (see e.g. Balas and Zemel, 1980; Hill and Reilly, 2000; Hall and Posner, 2007; Cho et al., 2008).
- For problems where the description is (partly) given by a matrix notation (like e.g. the city locations in a travelling salesman problem), several features regarding the dominance, sparsity, and density of the matrices have been shown to be good candidates (see e.g. Sassano, 1989; Vollmann and Buffa, 1966; Stützle and Fernandes, 2004). Alternatively, spectral properties of the matrices have also been used (Chung, 1997). When the matrix elements represent distances, it is useful to include features regarding the degree to which the triangle inequality is satisfied, and regarding the degree of clustering among the locations (see e.g. Cheeseman et al., 1991; Zhang and Korf, 1996; Ridge and Kudenko, 2007).

- For problems that are represented by graphs (e.g. graph colouring, covering, networking and routing), important features include the statistical properties of the graphs (node degrees), the size of cliques and spectral properties of the adjacency and Laplacian matrices of the graphs (see e.g. Chung, 1997; Eiben et al., 1998; White and Harary, 2001; Battiti and Protasi, 2001; Ou, 2005).

In their survey, Smith-Miles and Lopes conclude that there is little borrowing of concepts and ideas between different problem classes. Moreover, there are no clear rules on how good feature sets should be built. For practical optimisation problems, researchers usually reflect on the problem domain or try to find inspiration by speaking to human solvers familiar with the specific problem. While there exist feature descriptions for a limited number of problems, it remains impractical for researchers to go through the literature to find inspiration from other problem domains. In many cases, the details on the feature sets are even omitted, due to e.g. space restrictions. The relevant literature on this topic is furthermore distributed among several communities (machine learning, artificial intelligence and operational research among others).

For the problems considered in this dissertation (nurse rostering and project scheduling), there did not yet exist suitable domain-specific feature sets. It is part of this doctoral project to develop such sets and these will be discussed in detail in the corresponding chapters. Furthermore, in Chapter 5, we will discuss how such domain-specific feature sets can designed.

Domain-independent features

An interesting way of investigating the search space of problems is through the use of fitness landscape analysis (Reeves, 1999; Schiavinotto and Stützle, 2007). A fitness landscape is composed of a set of solutions, a fitness function assigning quality values to solutions, and a neighbourhood defining a distance metric on the solution space. A landscape can be conceived as a graph where the vertices are the solutions and edges indicate that two solutions lie within a predefined distance of each other. Building such a landscape could be considered domain-specific, as the solutions, the fitness function and the neighbourhoods are evidently related to the problem domain. The analysis of such landscapes however, is domain-independent and hence, several domain-independent features can be inferred from this landscape. Examples are the ruggedness (Angel and Zissimopoulos, 2000), the fitness-distance correlation (Jones and Forrest, 1995), the number and distribution of local optima (Weinberger, 1991), the structure of the basins of attraction (Bachelet, 1999), the degree of randomness in the location of the (local) optima (Locatelli and Wood, 2005), the density of the

states in the landscape (Rosé et al., 1996), and the Kolmogorov complexity of the landscape (which measures the degree of structure, as opposed to randomness) (Borenstein and Poli, 2006).

Many of these metrics have been employed for the characterisation of the complexity of several combinatorial optimisation problems and found useful for the prediction of problem difficulty. However, this relationship between landscape structure and problem difficulty is still poorly understood (Bierwirth et al., 2004). Moreover, note that a fitness landscape is not fully defined until all solutions to the problem instance are known. Only then, the complete landscape is known and the relevant features can be calculated. Such features are hence not suitable for an application where performance predictions need to be made without spending a lot of time calculating the possible solutions (like e.g. algorithm selection, which will be discussed in Section 2.3).

An alternative to fitness landscape analysis is based on the concept of landmarking (Pfahringer et al., 2000). The underlying idea is to run quick and/or simple algorithms for a limited amount of time on the problem instance, and to use information based on these runs as a source for identifying features. Such information could e.g. be the number of local optima encountered, or the evolution of the solution quality during a simple gradient descent local search run. The idea of landmarking is thus also domain-independent, but requires the availability of quick (domain-specific) solvers. The features resulting from landmarking can be either domain-specific or domain-independent.

An example: propositional satisfiability

As a prototypical example, we will now review a set of features for propositional satisfiability (SAT) problems. We have chosen this particular problem because it is a well-studied problem in artificial intelligence with many applications e.g. in software verification (Ivančić et al., 2005). Moreover, the features explained here will be applied later on in this dissertation in a totally different context.

Nudelman et al. (2004) introduce a set of 91 instance features for SAT problems. In their work on SATzilla (see Section 2.3), Xu et al. (2008, 2009) have extended this set to a total of 138 features, first publicly described in a technical report by Xu et al. (2012a) and only recently published by Hutter et al. (2014).

The features in the original paper (Nudelman et al., 2004) can be categorised into seven groups:

- *features related to the problem size:* (11)
  In SAT, problem size is measured by the number of variables v and the

number of clauses c. This group of features also contains the ratios c/v, v/c and the linearised ratio $|4.26 - c/v|$. Additionally, the second and third power of these ratios are included.

- *features based on the problem structure:* (24)
  SAT instances can be represented by different types of graphs: a variable graph, a clause graph and a bipartite variable-clause graph. The variable graph contains a node for every variable. An edge between two nodes indicates that both variables occur in a single clause. The clause graph contains a node for every clause. An edge between two nodes means that both clauses contain at least one common variable. The variable-clause graph is a bipartite graph with on one side a node for every variable, and on the other side a node for every clause in the problem. There are edges between clause nodes and variable nodes whenever the corresponding variable is contained in the respective clause. Various statistics (mean, variation coefficient, minimum, maximum and entropy) of the degree of the nodes in these graphs are used as features. For the clause graph, clustering coefficient statistics are also included.

- *features corresponding to the balance of the formula:* (13)
  These features include the ratio of positive (or negative) literals per clause and the ratio of positive (or negative) occurrences per variable. The fraction of unary, binary and ternary clauses is also included.

- *features measuring the proximity of an instance to a Horn formula:* (6)
  Horn formulae are a specific subset of general SAT problems being efficiently solvable (Chang and Lee, 1973). Features like the ratio of Horn clauses in the formula, and some statistics on the occurrence of variables in such clauses are included in the feature set.

- *features derived from solving linear programming relaxations of an integer model representing the instance:* (6)
  When the integer constraints are relaxed, linear programming instances become easier to solve. Specific characteristics of the solution to this relaxation, and the solution process (e.g. objective value, slack statistics, etc.) are considered.

- *features derived from running Davis-Putnam-Logemann-Loveland (DPLL) probes:* (7)
  This complete search algorithm (Davis et al., 1962) is repeatedly run with exponentially increasing depth. The features in this group are the number of unit propagations (for a set of different search depths), and the average depth at which a contradiction is found.

- *features based on local search probing:* (24)
  Multiple runs of two stochastic local search algorithms (SAPS (Hutter et al., 2002) and GSAT (Selman et al., 1992)) are performed. The selected features are related to the fraction of unsatisfied clauses, the number of steps to the best found solution, the average improvement along the path to the best found solution, and the variation of the number of unsatisfied clauses in each encountered (local) optimum.

Please note that the features from the latter two groups are closely related to the concept of landmarking discussed earlier in this chapter.

Hutter et al. (2014) have extended this feature set with three additional groups:

- *features derived from clause learning:* (18)
  A clause-learning SAT solver (Mahajan et al., 2005) is run for two seconds and various statistics on the number and size of the learned clauses are included as features.
- *features based on survey propagation:* (18)
  These features are derived from estimates of variable bias in a SAT formula, based on probabilistic inference (Hsu et al., 2008). Several measures of the probability that variables will be assigned either `true` or `false` are included.
- *features based on running time of the feature calculation process:* (12)
  The running times for calculating each (sub-) group of features are also included in the feature set.

The complete feature set thus includes both domain-specific and domain-independent features. Most domain-specific features can be easily calculated by simply counting values while iterating over all clauses and/or all variables. Features based on the concept of landmarking could possibly take more time. The required time for calculating the feature values may provide additional meta-information on the instances, which is why the final group of features is included in the set.

In the study of Nudelman et al. (2004), it is shown that the clauses-to-variables ratio is indeed important for the running time prediction of the considered complete search methods. Other important features are related to local search probing. It may initially be surprising that these features are informative for the running time of a complete solver. However, these features are actually related to the topology of the search space. They give an indication of the size and complexity of the search space, which also influences the running time of a

complete solver. We refer to Nudelman et al. (2004) for more details on these features.

2.2.2 Applying machine learning techniques

In Step 5 of the proposed procedure, it is advised to experiment with different machine learning techniques in order to find accurate prediction models. There are however many such techniques available. In the context of this dissertation, we do not want to go into details on the algorithmic design of such learning schemata. Instead, we are only interested in a purely functional application of such techniques. It is therefore not the aim of this section to give an overview of the available techniques. Instead, we want to discuss a number of practical considerations which are important when experimenting with such techniques constructing performance prediction models.

The field of machine learning is the sub-field of artificial intelligence concerning the development of computer programs being able to *learn*. Mitchell (1997) and Witten and Frank (2005) define learning in the following way:

> A computer is *learning* when it is changing its behaviour in a way that makes it perform better in the future.

This is an operational definition connecting learning to performance. A learning algorithm takes a number of training instances and outputs a model of *knowledge*. Closely related to machine learning is the field of data mining. Data mining can be defined as the process of (semi-) automatically discovering useful patterns in large quantities of data (Witten and Frank, 2005).

In this dissertation, we will use several techniques from the machine learning and data mining communities. Whenever useful, we will provide more details on the resulting models in the body of Chapters 4 and 5. We will apply these techniques in a purely practical sense, ignoring any implementation details or specific parameter settings. We will thus consider them as black boxes, using the default parameter settings. It will be our primary aim to build accurate performance prediction models, not to find the best parameter setting leading to the best prediction model, given a specific machine learning technique. During the doctoral project however, we did experiment with different parameter settings for a number of the best performing techniques, but as it turned out, this did not generally lead to significant improvements. Therefore, in the following chapters, we will report on the results using default parameter values, unless otherwise specified.

A toolbox allowing for such experimentations with different techniques, is the WEKA open source software tool of Hall et al. (2009). WEKA has been around for over fifteen years and has become an established toolbox for machine learning and data mining in practice. It contains state-of-the-art implementations of many commonly used techniques, and allows for user-friendly visual experimentation. We refer the reader to the comprehensive book of Witten and Frank (2005) for a detailed overview of the learning techniques present in WEKA.

In the next subsections, we discuss a number of important aspects to consider when applying machine learning techniques in the context of performance predictions.

Data preparation

The training data considered in this dissertation will generally consist of a large instance set, characterised by many features (up to over a few hundred different features). In practice, many of these values might be irrelevant for predicting the target value. Certain features will also be strongly (or even perfectly) correlated to other features, making them redundant. In general, and in particular in these cases, it is advised to prepare the data before trying to build prediction models.

An important part of data preparation concerns feature or attribute selection. For starters, uni-valued features are useless for learning purposes and could thus easily be removed. They could, however, expose a possible lack of variation in the instance distribution. Nevertheless, in any practical situation, the focus of the instance distribution will always be limited, leading to a number of uni-valued features that should be removed. Furthermore, in theory, the more attributes available, the better a learning method should perform. However, in practice, having too many (possibly irrelevant or redundant) attributes often confuses the learning methods. Another possible outcome is an over-fitted model. Such a model typically performs very well on the training data, but fails on new or unseen data because the noise in the training data has been modelled too. Hence, in the machine learning community, it is a common understanding that smaller models should be preferred over larger models. As an additional advantage, having fewer attributes also raises the potential for domain experts to interpret the models. It is easy to understand that a smaller decision tree allows for simpler explanations than a high-dimensional tree.

Attribute selection approaches can be categorised into two main groups: learner-independent techniques and learner-dependent techniques. The main difference is that learner-independent techniques select a feature set *prior to* the actual model construction, while learner-dependent techniques do attribute selection

while constructing the actual model. Learner-independent techniques can be based on simple data properties (e.g. on the result of an attribute correlation analysis) or on the outcome of other learners (e.g. on the coefficients of a simple linear regression model). Learner-dependent methods build an attribute set iteratively, using the actual model to find out which attributes lead to more accurate predictions. This can be done using either forward, backward or bi-directional selection. Forward selection starts with the attribute most correlated to the target value and iteratively adds more attributes to the model, until there is no more (significant) improvement on accuracy. Backward attribute selection is the reverse process and starts with a model containing many (or all) attributes and iteratively removes attributes until the accuracy starts to drop significantly. Bi-directional selection is a combination of both forward and backward selection.

There exist more ways for preparing raw data for machine learning purposes. Attribute discretisation, attribute transformation or data cleansing for example. In the context of this dissertation, these alternatives are less important. This is in contrast to many other applications, like e.g. biological applications, where human errors, faulty equipment and measurement noise pollute the data. In this dissertation, we have a stronger control over the data. We know that feature calculations will be correct, and that the algorithms will work according to the way they have been implemented. There are no missing values, and stochastic elements are treated by averaging the results over a number of independent runs. We will thus not need data preparation techniques other than attribute selection, and hence, we will not discuss such techniques in this section. We refer the interested reader to the book of Witten and Frank (2005) for a comprehensive discussion on these topics.

Evaluation of machine learning techniques

As briefly mentioned above, one of the main pitfalls in machine learning is the problem of over-fitting. Sometimes, the learned models are overly focused on the training data. In this case, extremely accurate predictions can be made for the training instances, but when the models are applied to new and unseen instances, the performance drops significantly. Attribute selection techniques can help in diminishing the problem of over-fitting. Nevertheless, it is not recommended to evaluate a model based on its performance on the training data. When possible, people should test their models on a separate instance set containing data that was not used for training. When large datasets are available, it is straightforward to randomly partition the complete dataset into a large training set and a decent-sized evaluation set. A model can then be learned on the training set and be evaluated on the evaluation set of unseen

instances. In order to reduce further over-confidence on the training set, we will apply 10-fold cross-validation on the training set for evaluating the learned prediction models. This process creates a 10-fold partitioning of the data and repeatedly learns a model leaving one fold out. These models are evaluated on the data that was left out, and the average performance over these 10 runs is reported. Cross-validation thus gives a better view on how well the learned model will perform on unseen data. We will thus compare different techniques based on their cross-validated performance. Once the best models have been selected, we will furthermore validate them on an additional set containing unseen data, in order to confirm their performance.

There exist different possibilities for measuring the performance of a prediction model. In order to evaluate classification algorithms, the easiest way is to look at the success rate of an algorithm, i.e. the number of correct predictions made by the model, with respect to the total number of instances. For numeric prediction models, different evaluation techniques are required. One could simply look at the average difference between the actual value and the prediction. A commonly used measure is the (root) mean squared error, as it considers the absolute value of the errors. However, the size of differences is often correlated to the size of the target value. It may thus sometimes be more useful to look at the relative differences. Another possibility is the coefficient of correlation (R) which measures the statistical correlation between the actual and the predicted target value. It ranges from -1 (perfect negative correlation), over 0 (no correlation at all) to 1 (perfect positive correlation). The advantage of a correlation coefficient is its scale independence. The question of what measure is best depends on the application, and is in general not easy to decide. Fortunately, in many practical situations, the best model is still the best, no matter what measure is used (Witten and Frank, 2005). In this dissertation, we will mainly use the correlation coefficient to select the best prediction models. When considering algorithm selection applications, we will also look at absolute and relative prediction errors.

2.3 Algorithm selection

As briefly introduced in Chapter 1, an important application of algorithm performance prediction models is found in algorithm selection systems. The idea of such algorithm portfolios has existed for many decades. One of the first formalisations of the algorithm selection problem was presented by Rice (1976). He identified the four key ingredients of the problem as follows:

- the problem space P

- the feature space F
- the algorithm space A
- the performance space Y

The problem space P is the set of problem instances of interest. The feature space F is identified with $\mathbb{R}^m$, the m-dimensional real vector space. There is a mapping $f(x) : P \to F$ characterising a problem instance $x \in P$ as an m-dimensional vector of feature values. The algorithm space A is the set of algorithms applicable to problem instances in P. The performance space Y identifies with $\mathbb{R}^n$, the n-dimensional real vector space of performance measures. The mapping $y(\alpha, x) : A \times P \to Y$ characterises the performance of algorithm $\alpha \in A$ on instance $x \in P$ as an n-dimensional vector of performance values. Let $||y(\alpha, x)||$ denote the norm of the performance vector $y(\alpha, x) \in Y$, providing one number to evaluate the performance of an algorithm α on a particular instance x. The algorithm selection problem can then be formally stated as follows:

> For a given problem instance $x \in P$ with feature values $f(x) \in F$, find a selection mapping $S(f(x)) : F \to A$ such that the selected algorithm $S(f(x)) = \alpha$ maximises the performance $||y(\alpha, x)||$. In other words, find a mapping $S(f(x))$ such that $||y(S(f(x)), x)|| \geq ||y(\beta, x)||$ for all $\beta \in A$.

While this framework contains all the necessary ingredients to build automatic algorithm selection tools, a number of these elements are formulated rather vaguely. It was already noted that the success of performance prediction models is highly dependent on the quality of the feature set. The same holds for algorithm selection systems. In order to find accurate mappings from instances onto algorithms, the instances must be adequately characterised by a set of features. The framework of Rice however, does not specify how such a set should be composed.

Apart from the features, the way the mapping $S(f(x))$ is constructed is also of crucial importance. There exist many different techniques in the machine learning community performing this task, each with their own strengths and weaknesses. Rice (1976) discussed several linear, piece-wise linear and polynomial approximation methods for building this mapping, but he already concluded that *"the determination of the proper non-linear form is still somewhat of an art and there is no algorithm for making the choice"*. Ironically, the choice of the best technique for building such mappings is in its turn also an algorithm selection problem. There is no single best machine learning algorithm to be applied for this task.

A straightforward way of building an algorithm selection strategy is through the use of empirical hardness models. In such a strategy, a set of empirical hardness models is built for a number of algorithms in a portfolio. When a new problem instance has to be solved, the performance of each algorithm is predicted and compared. The algorithm with the best predicted performance is selected to be run on the given instance. This is basically how the prize-winning portfolio solver SATzilla works. This portfolio solver was first introduced by Nudelman et al. (2004) and combines a number of publicly available, state-of-the-art SAT solvers. In the following years, the authors improved and extended their portfolio solver, leading to a series of medals in subsequent SAT competitions. The most recent description is presented by Xu et al. (2012b). The solver is actually more complicated than a simple algorithm selection tool. Up until the last version, SATzilla started by running a number of pre-solvers for a very short period of time. Doing so, a considerable number of instances in the competitions were already solved and no resources were wasted calculating features or evaluating prediction models. When the pre-solvers did not solve the instance in the given time, a number of features were calculated, and the running times of a set of algorithms were predicted. The best predicted algorithm was finally run. If feature calculation could not complete (due to an error or a time-out), a backup solver was chosen. This strategy won five medals in the 2007 SAT Competition,[3] and another five in the 2009 SAT Competition. The most recent version (SATzilla2012) won the 2012 SAT Challenge.[4] This version is no longer based on simple running time prediction models. Instead, cost-sensitive classification models are used. For each pair of solvers, a classification model is built for predicting which one of the two will perform better (i.e. solve an instance more quickly). When solving instances, at first, a classification model is used to predict whether or not feature calculation will be too costly. If feature calculation would be too costly, a backup solver is run. Otherwise, the pairwise models are used in combination with a voting system selecting the best solver for a given instance.

Smith-Miles (2009) gives a cross-disciplinary survey of meta-learning for algorithm selection. Meta-learning is conceived as learning about learning algorithms. The algorithm selection problem is considered as a meta-learning task, where the meta-data comprises the P, F, A and Y spaces, as defined above. Smith-Miles states that the key to a successful algorithm selection strategy lies within the F space. In other words, coming up with a qualitative feature set is crucial for the success of the meta-learning task. Smith-Miles further explicitly notes that it is not straightforward to design a set of suitable

[3] More information on the SAT Competitions can be found at http://www.satcompetition.org/.

[4] More information on the 2012 SAT Challenge can be found at http://baldur.iti.kit.edu/SAT-Challenge-2012/.

features for a given problem domain. In her survey, she discusses literature from a range of disciplines including machine learning, artificial intelligence, operational research, statistics, bio-informatics, etc. The literature is treated within the algorithm selection framework of Rice (1976). The common factors are the availability of large collections of problem instances, the existence of a number of diverse algorithms tackling the instances, a way of assessing their performance and the existence of a feature set adequately characterising the instances.

There have been several other algorithm selection approaches in the literature. Lobjois and Lemaître (1998) use performance prediction methods to select a branch-and-bound algorithm for constraint satisfaction problems. Vassilevska et al. (2006) use the term 'hybrid algorithm' for their algorithm selection approach to graph problems and constraint satisfaction problems. Pulina and Tacchella (2007, 2009) have built a so-called 'multi-engine solver' for quantified boolean formulas, which is inspired by, and hence follows the lines of the work on SATzilla. O'Mahony et al. (2008) use a case-based reasoning approach to build a portfolio solver for constraint solving. Musliu and Schwengerer (2013) have built an algorithm selection tool for graph colouring problems. Their portfolio consists of six state-of-the-art heuristic procedures. They have designed a feature set for graph colouring problems and use six different classification algorithms to build predictors based on these features. Their aim is not to predict running times, but they focus on predicting the algorithm choice directly. Their solver achieves a significantly better performance than any of the heuristics separately.

The previously mentioned algorithm selection approaches all treat the algorithms as black boxes. They make a choice a priori and run the chosen algorithm without further interference with the solution process. Alternatively, an algorithm selection approach could monitor the solution process and adapt its behaviour accordingly. One possibility is to *tune* algorithms based on instance properties. This approach is also called *automatic algorithm configuration* and will be discussed in Section 2.4. Another possibility is to include a set of heuristic building blocks in the portfolio, and to change the components of the running algorithm while it is solving an instance. This could be based on certain properties of the solving process. Such an approach could be placed under the concept of hyper-heuristics, which will be discussed in Section 2.5.

Gagliolo and Schmidhuber (2006) use the term *dynamic portfolio* to denote an algorithm selection technique which is not using an a priori learning phase. Instead, the candidate algorithms are run in parallel, and the running time distributions are learned while solving problem instances. After a while, based on running time predictions, fewer algorithms are considered to be candidates and computing power can thus be used more efficiently. However, due to the use of parallelism, this approach is inherently different from other algorithm

selection techniques. If a parallel infrastructure is not available, the computing power would have to be shared among the different solvers. This would result in a significant slowdown of the solution process.

Kotthoff (2012) focuses on which machine learning techniques to use when building algorithm selection tools for combinatorial optimisation problems. When such tools are based on running time predictions, Kotthoff recommends applying linear regression techniques. In the classification case (i.e. when the best algorithm should be predicted directly), he concludes that decision tree learners are among the most promising techniques. These results are not based on any theoretical ground, but are founded on empirical evidence using a number of constraint solvers on the different problems described in CSPLib (Gent and Walsh, 1999). Aiming at a more robust algorithm selection approach, Kotthoff proposed to use ensemble classification. In this approach, several classifiers are combined, and the final decision is based on a voting scheme. Doing so, the overall performance of an algorithm selection tool is further improved. The results of Kotthoff (2012) focus on constraint solving problems. In this dissertation, we consider other combinatorial optimisation problems that do not necessarily fit in the constraint solving framework. Our main focus is furthermore performance prediction. We will however consider the construction of algorithm selection tools as an application of performance prediction models. The work of Kotthoff (2012) is complementary to ours, but focuses mainly on algorithm selection in a constraint solving context.

2.3.1 Characterising algorithm sets for selection tools

In all publications cited above, the existence of a set of competitive algorithms is given a priori. There has not been any attempt at formalising the preconditions needed for any algorithm selection tool to be useful. Any relevant discussion on this topic currently lacks in the literature. In this subsection, we will introduce such an initial formalisation in the form of a novel framework characterising a set of algorithms in terms of their competitiveness and impact. This framework will be applied in Chapters 3 and 4, when attempting to construct automatic algorithm selection tools for the optimisation problems considered in this study.

As mentioned above, deciding on which algorithms to consider in an algorithm selection tool is mostly done in an ad-hoc manner. Here, we will present a number of properties to be satisfied such that the resulting tool may effectively improve over the best single-algorithm approach. The choice of algorithms is in fact arbitrary; but for an algorithm selection approach to be useful, the algorithms should be *competitive* on the considered instance set and their difference in performance should be *large*.

Below we introduce quantitative measures for the competitiveness and the potential impact of a set of algorithms on an instance set.

Competitiveness

We define the *competitiveness* among a set of algorithms in the following way:

> A set of algorithms is *competitive*, when each algorithm outperforms the other algorithm(s) on a subset of instances; while at the same time, it is outperformed by at least one other algorithm on other instances; and these subsets should furthermore be sufficiently large.

In case of two algorithms (here denoted as `algA` and `algB`), we define the following ratio as a measure for the competitiveness between the algorithms: Let A be the set of instances on which `algA` strictly outperforms `algB`, and B vice versa. Let T be the total set of instances. The *competitiveness ratio* c is defined as:

$$c = 2 \min(|A|/|T|, |B|/|T|) \tag{2.1}$$

The normalisation factor 2 is introduced to ensure that $0 \leq c \leq 1$.[5] This ratio denotes how competitive a set of two algorithms is. The lower c, the less competitive the algorithms are with respect to each other. This competitiveness depends on two factors: the *equipotency* of the algorithms and their *reach* in the instance set.

The *equipotency* e is defined as:

$$e = 2 \min(|A|/|A|+|B|, |B|/|A|+|B|) \tag{2.2}$$

The equipotency is a real number $0 \leq e \leq 1$. It reflects how evenly dominant the algorithms are over the other. If $e = 0$, this indicates that one algorithm is never performing worse than the other; if $e = 1$, this indicates that both algorithms each outperform the other one on an equal number of instances (i.e. that A and B are of equal size).

The *reach* r of a set of two algorithms is defined as:

$$r = |A|+|B|/|T| \tag{2.3}$$

The reach represents the fraction of the instances on which the algorithms perform differently. It thus gives an indication of the relative size of the subset

[5] This normalisation factor is not strictly necessary, but allows for a more natural interpretation using a range from 0 to 1.

of instances on which an algorithm selection tool can improve. The reach also equals 1 minus the fraction of instances on which both algorithms perform equally well.

The product of the equipotency and the reach equals the competitiveness ratio:

$$c = e.r \tag{2.4}$$

A high competitiveness ratio c thus indicates that the two algorithms differ on many instances, and that they are not unevenly dominated by one another. The competitiveness ratio of a set of algorithms also indicates how accurate an algorithm selection tool must be in order to produce a higher number of best choices than any single-algorithm strategy would. For highly competitive algorithms, the best single-algorithm strategy will be the best choice for a little more than half of the instances. For poorly competitive algorithms however, the best single-algorithm strategy will be the best choice for far more instances, up to nearly all instances. In fact, $1 - c/2$ is the fraction of best choices produced by the best single-algorithm strategy and hence a natural lower bound for the performance of any acceptable selection tool.

Potential impact

Competitiveness is not the only criterion for quantifying the usefulness of an algorithm selection tool. The actual difference in performance between both algorithms is also an important factor. Even for highly competitive algorithms, the difference in performance might be small, leading to only marginal performance improvements. Alternatively, consider the case where one algorithm is the best choice for most instances and the actual performance difference is small on these instances, while on the instances where the other algorithm is better, the differences are significantly larger. In such a situation, the best single-algorithm strategy may not be the algorithm that is best for most instances. Indeed, the overall absolute performance improvement on the small set of instances could outweigh the absolute performance improvement of the other algorithm on the larger set. When comparing algorithms, it is thus also important to look at the absolute performance improvements, apart from the number of wins.

Before formally introducing a factor reflecting the size of the actual performance differences, we need a little more notation. Let $m_{S,\texttt{algX}}$ denote the sum of the absolute performance achieved by `algX` on the set S of instances. This performance can e.g. be measured in terms of running times, or solution qualities. The total absolute performance of the optimal selection tool on the instance set

T can thus be calculated as follows:

$$m_{T,\texttt{opt}} = m_{A,\texttt{algA}} + m_{(T-A),\texttt{algB}} \tag{2.5}$$

Given this notation, we can introduce the *potential impact* i as:

$$i = \min\left(\frac{|m_{A,\texttt{algB}} - m_{A,\texttt{algA}}|}{m_{T,\texttt{opt}}}, \frac{|m_{B,\texttt{algA}} - m_{B,\texttt{algB}}|}{m_{T,\texttt{opt}}}\right) \tag{2.6}$$

The potential impact is the smallest of two (positive) values. For each algorithm the possible performance improvement is calculated and normalised to the optimal performance. The potential impact is thus a percentage indicating an upper bound for the relative overall performance improvement that could be realised by using an algorithm selection tool. It represents the average relative improvement that a perfect algorithm selection tool would achieve over the best single-algorithm strategy in a given instance set. When the potential impact is multiplied with the average performance over the instance set, an indication of the maximal average absolute performance improvement per instance is found.

The potential impact thus indicates the relative size of the possible performance improvements. A low potential impact can only lead to small relative performance improvements, while a high potential impact allows an accurate algorithm selection tool to result in large overall performance improvements. The product of the potential impact and the average absolute performance results in an indication of the maximal absolute performance improvement per instance.

Decision making

Whereas the competitiveness ratio indicates how accurate an algorithm selection tool must be in order to produce a higher number of best choices, the potential impact indicates how large the possible performance improvement of such a tool can be. The choice of algorithms to be considered in an automatic algorithm selection tool is thus dependent on both the competitiveness and the potential impact. Highly competitive algorithms can already benefit from moderately performing selection tools; while for poorly competitive algorithms, these tools should be highly accurate. Furthermore, the potential impact indicates the boundaries of the overall performance gains of such a tool. The potential impact can also be used to get an indication of the absolute performance improvements.

Currently, the definitions of these indicators are restricted to the case of two algorithms. In the case of more algorithms, these concepts require further generalisation. Pairwise comparison, leading to a set of such indicators, may provide a ranking, but is not informative on the individual contributions as

part of the whole. Another option is to compute indicators crosswise, denoting how competitive each algorithm is with respect to the combination of the other algorithms. The actual generalisation of these concepts is left as an opportunity for further research.

2.4 Algorithm configuration

The field of algorithm configuration is related to algorithm selection. An automatic algorithm configuration tool tries to configure or tune an algorithm, i.e. it tries to find a parameter setting (also referred to as a configuration) optimising the performance of an algorithm for a given problem instance, an instance set or a distribution.

This field has recently attracted considerable attention. Algorithms tend to have many parameters. Even for experts, tuning these parameters is a tedious task and often requires more time than the development of the algorithm itself. Furthermore, when applied in the real world, practitioners tend to use the preselected parameters presented in the research paper proposing the technique. These predetermined configurations may be tuned well for the benchmark problems considered in the respective paper, but when applied in practice, these configurations might lead to a degraded performance. Practitioners not familiar with the details of certain algorithms can find it a very difficult task to tune such algorithms for their own specific setting (See e.g., Hutter et al., 2010).

Nowadays, several frameworks exist allowing the automatic configuration of algorithms, regardless of whether the parameters are numerical (continuous or discrete), ordinal (e.g. low/medium/high) or categorical. Categorical parameters can also include discrete building blocks for higher-level (meta-) heuristics. In that sense, automatically choosing the building blocks of a heuristic algorithm can also be considered in the hyper-heuristic framework. We will discuss this topic in Section 2.5.

Minton (1996) presents the MULTI-TAC system. This system automatically selects the most promising configuration for a generic heuristic. It enumerates possible configurations and selects the best one using a hill-climbing technique.

Adenso-Diaz and Laguna (2006) introduce the CALIBRA framework allowing the automatic tuning of search parameters by using a fractional factorial experimental design, combined with a gradient-descent local search method. The results are limited by the fact that the strategy can only handle up to five numerical or ordinal parameters (thus no categorical parameters). Moreover, possible interaction effects of the parameters are ignored.

Preuss and Bartz-Beielstein (2007) present the concept of sequential parameter optimisation. It first runs an algorithm on a set of training instances, using parameter configurations from a factorial experimental design. It then fits a response surface model on the outcome. The predictions of the model are used to determine the next configuration to be evaluated. More recently, Hutter et al. (2009a, 2011) improved this framework.

Hutter et al. (2007, 2009b) present the ParamILS framework for automatic algorithm configuration. Their aim is to optimise an algorithm's performance on a class of problem instances. This goal is achieved through the search for optimal settings for numerical, ordinal and categorical parameters. Good parameter configurations are searched for using iterated local search (Lourenço et al., 2002). The effectiveness of their approach is demonstrated using the commercial solver IBM ILOG CPLEX on a variety of mixed integer programs.

Another approach is based on racing algorithms, as developed in the machine learning community (Maron and Moore, 1997). Birattari et al. (2002) and Birattari (2004) introduce the F-race procedure and use it to configure several stochastic local search algorithms. Further development of these ideas has led to the irace package of López-Ibánez et al. (2011). This package implements an iterated version of the racing procedure to find an appropriate configuration of an optimisation algorithm for a given set of instances.

Kadioglu et al. (2010) introduced the ISAC framework. The acronym stands for Instance-Specific Algorithm Configuration. In contrast to the previous examples, the ISAC framework does not aim at finding a good parameter configuration for an instance set or distribution. Instead, the goal is to find good configurations for individual instances. The training data is first clustered, after which the automatic algorithm configurator GGA (Ansótegui et al., 2009) is used to find good parameter settings for each cluster. When given a new instance, the framework first determines the cluster to which it belongs and then runs the algorithm with the corresponding parameter setting.

Some researchers have combined automatic algorithm selection and algorithm configuration techniques. The Hydra framework of Xu et al. (2010) is an example of such a combination. Hydra aims at building a set of algorithms with complementary strengths, using the FocusedILS configurator of Hutter et al. (2009b). The automatic algorithm selection technique being used is similar to the SATzilla framework described earlier. Hydra starts with a portfolio containing only one configuration of a certain algorithm and iteratively adds other configurations when they can improve on the performance of the current portfolio. Some (older) configurations can also be dropped from the portfolio depending on the overall performance. The success of this approach is demonstrated on local search heuristics for the SAT problem (Xu et al., 2010).

Xu et al. (2011) have further improved the HYDRA framework and applied it to mixed integer programming, resulting in the HYDRA-MIP framework. This version converges quicker and uses classification tools instead of linear regression for the algorithm selection part.

2.5 Hyper-heuristics

Recently, hyper-heuristics have drawn significant attention (See e.g. the PhD dissertation of Misir, 2012). The term *hyper-heuristic* was first used by Denzinger et al. (1997). It was first introduced in a journal paper by Burke et al. (2003). It can be understood as a method operating on a set of low-level heuristics. There is a domain barrier present between the actual problem and the hyper-heuristic procedure. The hyper-heuristic itself is unaware of the actual problem being solved, it only operates on the low-level heuristics. The low-level heuristics in their turn build or improve solutions for the actual problem instance. The hyper-heuristic coordinates the solution process, deciding which low-level heuristic should be applied next. This explains the main difference with metaheuristics. Metaheuristics guide a search from one solution to another, applying different heuristic techniques. They thus operate on a search space of solutions. Hyper-heuristics are only concerned with the low-level heuristics, regardless of the actual solutions (i.e. the actual problem domain).

One possibility for a hyper-heuristic to arrive at such a management capability is through the implementation of machine learning techniques allowing for the prediction of the behaviour of each low-level heuristic in any eventual state of the process. Recently, state-of-the-art machine learning techniques have been proven very efficient in this respect (Misir, 2012).

In the literature, a distinction is made between two main types of hyper-heuristics: *selection hyper-heuristics* and *generation hyper-heuristics*. Selection hyper-heuristics are heuristics to *choose* heuristics, while generation hyper-heuristics are heuristics to *build* heuristics. Selection hyper-heuristics are conceptually similar to algorithm selection techniques. The set of low-level heuristics is predetermined and the hyper-heuristic merely coordinates the search process by selecting the algorithm to be applied, regardless of the actual problem domain. Generation hyper-heuristics are different. They build up a solution strategy from a set of basic building blocks. This relates to algorithm configuration in case the parameter configurations include choosing the algorithmic components to be employed.

We refer to Burke et al. (2010) for a more extensive classification of hyper-heuristic approaches.

2.6 Algorithm footprints

A concept closely related to empirical hardness and performance prediction is the notion of an *algorithm footprint*.

Corne and Reynolds (2010) observe that many researchers in optimisation publish new techniques or algorithms, based on their state-of-the-art performance on certain, well-chosen instance sets or benchmarks. Doing so, parameters are tuned (either manually or automatically) in order to optimise the average performance over the considered instances. Algorithms are compared based on this average performance, and conclusions are drawn stating that an algorithm is better, at least as good, or not significantly worse than the current state of the art. Corne and Reynolds (2010) note, however, that there exist many coherent subsets of the instance distribution (e.g. based on the values of some instance parameters) on which other algorithms (or other parameter configurations for a certain algorithm) can outperform the overall best algorithm (or parameter configuration). The overall best choice for a distribution is thus not necessarily the best choice for each subset of that distribution.

Corne and Reynolds (2010) stress the importance of specifying the boundaries of algorithm performance in instance space. They introduce the term *algorithm footprint* as a means to indicate how an algorithm's performance generalises in instance space. The idea is to delimit the region in the instance space where an algorithm is performing well. In their empirical study, the instance sets are partitioned into clear and well-defined regions based on simple characteristics of the instances. They study the performance of a large set of algorithms (or configurations of algorithms) on these regions and demonstrate that the overall best algorithm is rarely the best algorithm for all subsets. They have studied two problem domains in particular: single-machine job-shop scheduling and vehicle routing. While this is clearly an important observation, their approach has some limitations. First of all, the instance sets are partitioned a priori. This requires domain knowledge and specific insight into the correlation between instance properties and algorithm performance. Second, it is not investigated whether this ad-hoc partitioning is the only possible partitioning, nor what the best level of granularity is. It might very well be the case that the overall best algorithm for one predefined cluster is not the best algorithm for each individual instance within this cluster. While well-defined clusters can help to understand the relationship between instance properties and algorithm performance, it is not clear whether the real algorithm footprints are defined in these terms. Footprints might actually look a lot more capricious.

In this dissertation, we focus on the per-instance performance of algorithms, which is a more fine-grained approach than the footprints of Corne and Reynolds

(2010). There thus exists a trade-off between the finer granularity (i.e. more accurate modelling) and the simplicity of the relationship between instance properties and algorithm performance and hence the potential of a simple and clear explanation.

Related to these footprints is the work of Smith-Miles and Lopes (2011) and Smith-Miles and Tan (2012). Here, the concept of an algorithm footprint is used to delimit the region where the algorithm yields good performance. Their approach however differs from the one of Corne and Reynolds (2010). Smith-Miles and Lopes (2011) also notice the drawbacks of the a-priori splitting of the instance space based on one or two simple features. Instead, they use self-organising feature maps to visualise how algorithm performance varies over the instance distribution. Self-organising maps were developed by Kohonen (1982) as a way to automatically detect strong features in large datasets. The technique employs artificial neural networks mapping high-dimensional feature spaces onto two-dimensional maps. Smith-Miles and Lopes (2011) focus on a timetabling problem used in the 2007 International Timetabling Competition (ITC2007). The instances under study are a limited number of real-world instances from Udine University, together with two sets of generated real-world-like instances. Two algorithms, ranked top-5 in the competition, are compared. The instances are characterised by 32 features. Smith-Miles and Lopes show that self-organising maps can clearly distinguish between the three types of instances (real-world, or element of one of the two generated sets). However, when superimposing the algorithm performance onto these maps, and hence revealing the footprints of the algorithms, the boundaries of the regions are far less clear. Both algorithms are competitive across the two-dimensional map. In large portions of the map, there is no clear difference between the algorithms. There are only a few smaller regions where the algorithms do differentiate. However, in these regions, there is still no clear winner. There is only one narrow line going through the map on which one algorithm is consistently better than the other. It is however very difficult to describe this line in terms of feature values. The self-organising maps may thus provide a visual representation of the instance space with clear distinctions between the different types of instances. However, when looking at the algorithm footprints, the delimitations of these regions are far less clear. Even for regions where there is a clear winner, there is no straightforward explanation for this.

Smith-Miles and Tan (2012) go one step further in formalising a methodology for defining, visualising and characterising an algorithm footprint across a high-dimensional space. They again note that a necessary precondition is the availability of a qualitative feature set relating to algorithm performance. Smith-Miles and Tan claim that for many optimisation problems, there is already much known about the properties that make these problems easy or hard. While this

may be the case for a number of prototypical problems, it is certainly not the case for many other practical problems. As discussed in Section 2.2.1, there is no general or straightforward way of building an adequate feature set for any given problem. There exist a number of studies on the intrinsic hardness of several problems, revealing certain properties related to phase transitions or easy-hard-easy patterns in solving these problems. It is however not clear whether these properties also relate to the empirical hardness of the problems, or in other words to the performance of certain algorithms solving these problems. We could thus state that the existence of an adequate feature set is not at all evident.

Smith-Miles and Tan (2012) discuss a method for building visual representations of algorithm footprints in case an adequate feature set exists. In order to have a clear view on the performance of the considered algorithms, they construct the instance sets so that they intentionally contain especially hard and easy instances for the considered algorithms. They use evolutionary algorithms to evolve a set of such instances (Smith-Miles et al., 2010; Smith-Miles and van Hemert, 2011) and perform a principal component analysis projecting the high-dimensional feature space onto a two- or three-dimensional space (defined by the two or three most dominant principal components). As in the work of Smith-Miles and Lopes (2011), the instances are visualised in a two- or three-dimensional map, but now based on the principal component analysis instead of using a self-organising map. Algorithm performance is further on superimposed on each instance, revealing the footprints of several algorithms. Because of the way the dataset is composed, the regions where an algorithm performs well/bad are clearly visible. The quality of an algorithm can further be assessed by calculating the area of the convex hull of the data points (i.e. instances) for which the algorithm performs well. This also allows for a comparison between algorithms. However, care must be taken when drawing conclusions. Note that such an area is highly dependent on the principal components, and hence on the quality of the feature set. Furthermore, the evolved instance sets do not necessarily cover all regions where an algorithm might perform good or bad. While these approaches are clearly a step towards a better visualisation of algorithm performance, there is still a lot more work to be done towards the understanding of the relationship between instance features and algorithm performance.

2.7 Conclusions

The aim of this chapter was to build a context for the doctoral project presented in this dissertation. We did this by reviewing a number of key publications in

the relevant field, and by proposing novel ideas where necessary.

We started this chapter with a discussion on one of the most important concepts for this research: *empirical hardness*. The term refers to the apparent complexity of a problem instance, as it is observed by an algorithm. We stressed that the two keywords of the term, (*empirical* and *hardness*), are equally important.

The second section focused on the construction of empirical hardness models. We proposed a generalisation of an existing strategy for building such models. This generalisation allows for a much broader field of applications. The most important adaptation concerns the incorporation of deciding on a performance criterion. Doing so, the strategy can essentially be applied to any computational problem being solved by any algorithm (complete or incomplete). The proposed procedure is formulated at a high level. For it to be applied in a specific setting, the ingredients need to be carefully instantiated. We have dedicated a subsection on the selection of a valuable feature set. There is a growing interest in this topic and consequently, there are a number of problem domains for which good features exist. We include a discussion of an example feature set for a well-known problem domain. However, the literature does not cover a wide area of problems yet. In this dissertation, we complement the existing literature on this topic by designing extensive feature sets for two additional problem domains: nurse rostering and project scheduling. Another subsection is dedicated to the application of machine learning techniques in this context. We stress the importance of data preparation and present a brief discussion on how to evaluate the resulting models.

One of the main applications of performance prediction models is found in automatic algorithm selection tools. We discussed the theoretical framework of Rice (1976) on this topic and refer to a number of developments in this direction. We furthermore stressed the lack of any theoretical basis for deciding which algorithms to include in such an application and proposed a novel framework supporting this choice. This framework is based on the concepts of competitiveness and potential impact.

We briefly discussed the related problem of automatic algorithm configuration and touched the recently emerged field of algorithm footprints. One of the main aims there is to find and describe the boundaries of good algorithm performance and to express these in terms of feature values. The work in this field can help to understand why certain algorithms perform well on certain instances and not on other instances. However, this research area is still rather young, and there is a lot more work to be done before such valuable explanations can be made. In this dissertation, we will show how empirical hardness models can play a supportive role in understanding algorithm behaviour.

Chapter 3

Nurse rostering

One of the key challenges for the well-functioning of large companies is the efficient management of its human resources. This is especially the case for health care organisations, where the number one priority is delivering a qualitative service. The quality and the motivation of the human staff are among the critical success factors. Everything starts with the acquisition of the appropriate staff. It is furthermore crucial for the well-functioning of a hospital ward that this staff is well supported. Attention for the personal well-being of the staff should not be underestimated. Such support can be achieved by taking personal preferences into account, e.g. allowing nurses to work the specific shifts they requested, or considering their preference to be working in teams with certain other co-workers. Unattractive schedules, poor environments and high workloads generally lead to a poor service quality level. A proper management of the workforce has a positive effect on the working conditions. These working conditions are in their turn strongly related to the service quality level of health care and the recruiting of qualified personnel (Berrada et al., 1996; Burke et al., 2004; Bard and Purnomo, 2005).

In this chapter, we will focus on nurse rostering problems. We will consider a fixed set of nurses who must be assigned to a set of shifts during some scheduling period in a hospital ward. The assignment must satisfy a number of predetermined coverage demands, while minimising the violations of a set of (possibly conflicting) constraints. We start with a literature overview in Section 3.1, after which, in Section 3.2, a clear definition and a mathematical model of the specific problem considered in this chapter is presented. In Section 3.3, we elaborate on the experimental study concerning algorithm performance predictions for nurse rostering. Section 3.4 deals with possible applications

and in Section 3.5, we draw conclusions from this chapter and point towards possible extensions.

3.1 Literature overview

It is not the aim of this section to give an extensive overview of the literature on solving specific nurse rostering problems. We rather want to sketch the ingredients of such problems and want to show their relevance in today's world.

Smith et al. (1979) describe the nurse workforce management process as the combination of three separate and sequential tasks: (1) staffing, (2) scheduling and (3) allocation. The staffing task addresses the acquisition of a qualitative nurse pool. This includes the recruitment of nursing staff with specific characteristics (like full- or part-time contracts, specific skills and complementary shift preferences) such that the expected workloads can be covered. During the second phase, a schedule is built such that the coverage requirements are met, while trying to satisfy a number of additional constraints. The last phase deals with unforeseen circumstances, like patient overloads or absent nurses, and is thus performed while the generated schedules are employed in practice. It is easy to understand that each stage restricts the following stage. In this chapter, we are interested primarily in the second phase and denote this as the nurse rostering problem. The nurse rostering problem is actually a subclass of more general personnel scheduling problems encountered in many different application areas. One thing they all have in common is the fact that it is of crucial importance to have the right staff at the right place for the job. In general, a personnel scheduling problem deals with constructing timetables or rosters for a fixed workforce, given the predetermined coverage requirements, while satisfying a number of additional constraints. Every domain has its own specific characteristics, leading to a variety of different models and solution techniques. In general, we can state that personnel scheduling problems are complex and highly constrained optimisation problems (Glover and McMillan, 1986; Ernst et al., 2004).

Throughout the years, many different formulations have been proposed for personnel scheduling problems in general, and for nurse rostering problems in particular. A model is always some abstraction of the actual real-world problem, where certain aspects are considered more important than others. Eventually, there are always some details left out of the scope of the specific paper. There is thus no generally accepted or commonly used model in the literature. In fact, this large diversity in research papers closely resembles the real-world situation. Hospitals are organised differently in different countries.

Even within one country, each hospital has its own specific problem formulation. It is furthermore the case that within one hospital, the several departments focus on different aspects. This results in a wide variety of problems, all under the same name of *nurse rostering*.

Given this situation, it is not easy to compare different problems, instances or solution methods. Moreover, it is almost impossible to straightforwardly apply existing methods to (new) problems or instances with just even a slightly different formulation. De Causmaecker and Vanden Berghe (2011) have undertaken a thorough effort coming up with a unified framework in which researchers can position their work. The notation facilitates the problem description and classification. It enables researchers to position their work in the broad literature. We refer the reader to this paper for a general overview of nurse rostering problem formulations. In this chapter, we will introduce one specific formulation for the problem, which we will use throughout this dissertation. Whenever we speak of the nurse rostering problem, it will refer to this specific formulation, unless explicitly denoted otherwise.

Nurse rostering is a hard optimisation problem. Osogami and Imai (2000) have proven that it is in general NP-hard. They have actually proven that the timetabling problem, which is NP-complete, can be transformed into a decision version of a nurse rostering problem. The resulting problem furthermore considers only a subset of the real-world constraints applying to it. Recently, Smet et al. (2014) have shown that specific subclasses of nurse rostering problems are polynomially solvable when represented using a network flow formulation. However, the restrictions are very limiting. Most instances considered in this dissertation do not fall within these classes.

3.2 Problem definition

The nurse rostering problem (NRP) considered in this study is employing the formulation of the First International Nurse Rostering Competition (2010 INRC, see Haspeslagh et al., 2012). The choice for this model is a natural consequence of the fact that our research group co-organised the competition for which it was introduced. Some experience with this model was thus already present. We will first introduce the important ingredients, after which we will present a mathematical description of the model in Section 3.2.1.

The nurse rostering problem is essentially the problem of assigning nurses to a set of shifts that have to be worked. The difficulty of the problem lies in the constraints that have to be satisfied. Some constraints are considered hard while others are considered soft. For a solution to be *feasible*, all hard constraints

must be satisfied. An *optimal* solution is a feasible solution that satisfies as many soft constraints as possible. Soft constraints can optionally be weighted. In that case, an optimal solution minimises the weighted sum of constraint violations. More specifically, a nurse rostering problem consists of the following ingredients:

- *a scheduling horizon:* This is a number of consecutive days during which the nurses need to be assigned to shifts. The scheduling horizon can be of arbitrary size.
- *shift types:* Each day is divided into a number of (possibly overlapping) shift types. A shift type represents a time frame and specifies a skill set. (Nurses are required to have specific skills to be able to work certain shift types.[1])
- *coverage constraints:* These are the personnel requirements. For each shift type on each day, the number of required nurses is specified.
- *a nurse pool:* This is the set of nurses of the hospital ward. Nurses have their own characteristics which are described by their skill set, their contract regulations and their specific requests.
- *skill sets:* Each nurse can have multiple skills. Nurses are allowed to work shift types for which they own all the required skill types.
- *contracts:* Each nurse works according to the regulations specified in exactly one contract. A contract specifies the constraints (and the associated weights) imposed on the personal schedules of the nurses. Each contract specifies the following constraints:
  - maximum and minimum number of shift assignments during the scheduling period
  - maximum and minimum number of consecutive working days
  - maximum and minimum number of consecutive free days
  - maximum and minimum number of consecutive working weekends
  - maximum number of working weekends allowed per four weeks
  - whether or not the nurse should have at least two days off after a series of night shifts
  - whether or not the working weekends should be complete (i.e. if the working weekends should be complete, then either a nurse works full weekends or not at all during the weekend)

[1] For example: there might be a `Day` shift type that requires a regular nurse, and a `DayHead` shift type that requires a head nurse.

  - whether or not the nurse should work identical complete weekends (i.e. whether or not he or she should work the same shifts on all days within a weekend)
  - a set of unwanted patterns (e.g. a nurse is not allowed to work an early shift directly after a late or night shift)

- *individual requests:* A nurse can request to work or to be free on a particular shift on a certain day, or even any shift on a particular day.
- *global constraints:* These constraints are not individual, but set for all nurses in the pool.
  - single assignment constraint: This is a global constraint that specifies whether or not nurses should work at most one shift type per day.
  - alternative skill: This constraint specifies whether or not nurses are entitled to work shift types that require skill types for which they do not qualify.

Patterns can be of any form. In the 2010 INRC, two types of patterns were present. The first pattern type specifies a series of consecutive shift type assignments that should always be avoided (i.e. the pattern can start at any day in the scheduling horizon). The second pattern type specifies that a nurse should not have a free day before working any of a number of consecutive days (e.g. when a nurse has to work either on Saturday or Sunday, then she should also work on Friday). Other arbitrary patterns are possible, but are not considered in this dissertation.

This model has recently been generalised by Smet et al. (2012). The work presented in this chapter, however, was carried out before the publication of this generalisation.

3.2.1 Mathematical model

We present an exact description of the nurse rostering problem as a mixed integer program. This is basically the same model as presented by Bilgin (2012) and Haspeslagh (2012), which was developed in a joint-work collaboration (Messelis et al., 2009).

Symbols:		
N	:	The set of nurses n.
D	:	The set of days d.
W	:	The set of weekends w.
S	:	The set of shift types s.
SK	:	The set of skill types sk.
SK_s	:	The set of skill types sk required for shift type s.
SK_n	:	The set of skill types sk of nurse n.
$D^{on/off}$	:	The set of requests $req_{n,d}^{on/off}$ denoting whether nurse n does/does not want to work on day d.
$S^{on/off}$	:	The set of requests $req_{n,d,s}^{on/off}$ denoting whether nurse n does/does not want to work shift type s on day d.
PAT_n^t	:	The set of unwanted patterns of type t for nurse n. ($t \in \{cons, free\}$, see below)

Constraints:		
$C_{d,s}$	:	The number of nurses required to work shift type s on day d.
$T_n^{max/min}$	:	The maximum/minimum total workload of nurse n, measured by the number of shifts he/she is working.
$CW_n^{max/min}$	:	The maximum/minimum number of consecutive working days for nurse n.
$CF_n^{max/min}$	:	The maximum/minimum number of consecutive free days for nurse n.
$CWW_n^{max/min}$	:	The maximum/minimum number of consecutive working weekends for nurse n.
CW_n	:	Boolean constraint (value 1 or 0) indicating whether nurse n should be working complete weekends.
CIW_n	:	Boolean constraint (value 1 or 0) indicating whether nurse n should be working identical weekends.
SK^{alt}	:	Global boolean constraint (value 1 or 0) indicating whether nurses are allowed to work shift types for which they do not have the right skill set.

Table 3.1: Symbols and constraints for the mathematical model of the NRP.

We introduce the notation in Table 3.1. Expression (3.1) defines the binary decision variables $x_{n,d,s}$. They state whether a nurse n is working shift type s on day d.

$$\forall n \in N, \forall d \in D, \forall s \in S : \\ x_{n,d,s} = \begin{cases} 1 \text{ if nurse } n \text{ is assigned a shift type } s \text{ on day } d \\ 0 \text{ otherwise} \end{cases} \tag{3.1}$$

A set of auxiliary variables $p_{n,d}$ is defined in Expression (3.2). These variables indicate whether nurse n is working any shift type on day d. A similar set of auxiliary variables $q_{n,w}$ is defined in Expression (3.3) indicating whether a nurse n is working any shift during weekend w. A weekend w is defined by its first day $d_{w,1}$ and its length k (which is equal for all weekends in the planning horizon).[2]

$$\begin{aligned} \forall n \in N, \forall d \in D : -|S|.p_{n,d} + \sum_{s \in S} x_{n,d,s} &\leq 0 \\ \text{and } -p_{n,d} + \sum_{s \in S} x_{n,d,s} &\geq 0 \end{aligned} \tag{3.2}$$

$$\begin{aligned} \forall n \in N, \forall w \in W : -k.q_{n,w} + \sum_{i=0}^{k-1} p_{n,d_{w,1+i}} &\leq 0 \\ \text{and } -q_{n,w} + \sum_{i=0}^{k-1} p_{n,d_{w,1+i}} &\geq 0 \end{aligned} \tag{3.3}$$

There are two types of hard constraints. Expression (3.4) defines the coverage requirement constraints (enforced by $C_{d,s}$). Expression (3.5) defines the single assignment per day constraint, stating that nurses should work at most one shift type per day.

$$\forall d \in D, \forall s \in S : \sum_{n \in N} x_{n,d,s} = C_{d,s} \tag{3.4}$$

$$\forall n \in N, \forall d \in D : \sum_{s \in S} x_{n,d,s} \leq 1 \tag{3.5}$$

[2] for example: a weekend w of $k = 2$ days consists of the consecutive days $d_{w,1}$ and $d_{w,2}$.

The other constraints in the problem are all considered soft. These soft constraints can be weighted for each nurse individually. Each nurse n has therefore a weight W_n^C associated with each constraint type C. When violated, these constraints contribute to the objective function proportionally to the degree of violation, multiplied by the individual weight of the corresponding nurse. The objective function is given in Expression (3.6). The goal is to minimise a sum of weighted constraint violations, defined by the expressions (3.7 - 3.31).

$$\begin{aligned} \text{MINIMISE}\{\quad & V(T^{max}) + V(T^{min}) + V(CW^{max}) + V(CW^{min}) + \\ & V(CF^{max}) + V(CF^{min}) + V(CWW^{max}) + V(CWW^{min}) + \\ & V(CW) + V(CIW) + V(D^{off}) + V(S^{off}) + V(D^{on}) + \\ & V(S^{on}) + V(SK) + V(PAT^{cons}) + V(PAT^{free}) \qquad \} \end{aligned} \tag{3.6}$$

Constraints T_n^{max} and T_n^{min} on the total workload of nurses limit respectively the maximum and minimum number of shifts worked per nurse n. The total number of weighted constraint violations $V(T^{max})$ and $V(T^{min})$ are defined in Expressions (3.7) and (3.8).

$$V(T^{max}) = \sum_{n \in N} W_n^{T^{max}} max\{(\sum_{d \in D} \sum_{s \in S} x_{n,d,s}) - T_n^{max}, 0\} \tag{3.7}$$

$$V(T^{min}) = \sum_{n \in N} W_n^{T^{min}} max\{T_n^{min} - (\sum_{d \in D} \sum_{s \in S} x_{n,d,s}), 0\} \tag{3.8}$$

The constraints CW_n^{max} and CW_n^{min} limit the maximum and minimum number of consecutive working days for each nurse n. The total number of weighted constraint violations $V(CW^{max})$ and $V(CW^{min})$ are defined in Expressions (3.9) and (3.10), subject to the inequalities (3.11) and (3.12). The auxiliary variables $t_{n,d,0}$ in (3.11) indicate whether a nurse n works on day $d+1$ while he or she is free on day d. For $i > 0$, the auxiliary variables $t_{n,d,i}$ in (3.12) indicate whether there is a consecutive row of working days (of length i) for nurse n

since day d. In counting these constraint violations, nurses are assumed to be free on days outside the planning period (i.e. $p_{n,\alpha} = 0$ for $\alpha \notin D$).

$$V(CW^{max}) = \sum_{n \in N} W_n^{CW^{max}} \sum_{d \in D} max\{(\sum_{i=0}^{CW_n^{max}} p_{n,d+i}) - CW_n^{max}, 0\} \quad (3.9)$$

$$V(CW^{min}) = \sum_{n \in N} W_n^{CW^{min}} \sum_{d \in D} (CW_n^{min}.t_{n,d,0} - \sum_{i=1}^{CW_n^{min}} t_{n,d,i}) \quad (3.10)$$

$$0 \leq t_{n,d,0} \leq 1 - p_{n,d} \text{ and } p_{n,d+1} - p_{n,d} \leq t_{n,d,0} \leq p_{n,d+1} \quad (3.11)$$

$$0 \leq t_{n,d,i} \leq p_{n,d+i} \text{ and } t_{n,d,i} \leq t_{n,d,i-1} \quad (3.12)$$

Similarly, the constraints CF_n^{max} and CF_n^{min} limit the maximum and minimum number of consecutive free days for each nurse n. The total number of weighted constraint violations $V(CF^{max})$ and $V(CF^{min})$ are defined in Expressions (3.13) and (3.14), subject to the inequalities (3.15) and (3.16). The auxiliary variables $t_{n,d,0}$ in (3.15) indicate whether a nurse n works on day d while he or she is free on day $d+1$. For $i > 0$, the auxiliary variables $t_{n,d,i}$ in (3.16) indicate whether there is a consecutive row of free days (of length i) for nurse n since day d. In counting these constraint violations, nurses are assumed to be working on days outside the planning period (i.e. $p_{n,\alpha} = 1$ for $\alpha \notin D$).

$$V(CF^{max}) = \sum_{n \in N} W_n^{CF^{max}} \sum_{d \in D} max\{(\sum_{i=0}^{CF_n^{max}} 1 - p_{n,d+i}) - CF_n^{max}, 0\} \quad (3.13)$$

$$V(CF^{min}) = \sum_{n \in N} W_n^{CF^{min}} \sum_{d \in D} (CF_n^{min}.t_{n,d,0} - \sum_{i=1}^{CF_n^{min}} t_{n,d,i}) \quad (3.14)$$

$$0 \leq t_{n,d,0} \leq p_{n,d} \text{ and } p_{n,d} - p_{n,d+1} \leq t_{n,d,0} \leq 1 - p_{n,d+1} \quad (3.15)$$

$$0 \leq t_{n,d,i} \leq 1 - p_{n,d+i} \text{ and } t_{n,d,i} \leq t_{n,d,i-1} \quad (3.16)$$

The third type of consecutiveness constraints, CWW_n^{max} and CWW_n^{min}, limit respectively the maximum and minimum number of consecutive working weekends per nurse n. The total number of weighted constraint violations $V(CWW^{max})$ and $V(CWW^{min})$ are defined in Expressions (3.17) and (3.18), subject to the inequalities (3.19) and (3.20). The auxiliary variables $t_{n,w,0}$ in (3.19) indicate whether a nurse n works on weekend $w+1$ while he or she is free on weekend w. For $i > 0$, the auxiliary variables $t_{n,w,i}$ in (3.20) indicate whether there is a consecutive row of working weekends (of length i) for nurse n since weekend w. In counting these constraint violations, nurses are assumed to be free on weekends outside the planning period (i.e. $q_{n,\alpha} = 0$ for $\alpha \notin W$).

$$V(CWW^{max}) = \sum_{n \in N} W_n^{CWW^{max}} \sum_{w \in W} max\{(\sum_{i=0}^{CWW_n^{max}} q_{n,w+i}) - CWW_n^{max}, 0\} \tag{3.17}$$

$$V(CWW^{min}) = \sum_{n \in N} W_n^{CWW^{min}} \sum_{w \in W} (CWW_n^{min}.t_{n,w,0} - \sum_{i=1}^{CWW_n^{min}} t_{n,w,i}) \tag{3.18}$$

$$0 \leq t_{n,w,0} \leq q_{n,w} \text{ and } q_{n,w} - q_{n,w+1} \leq t_{n,w,0} \leq 1 - q_{n,w+1} \tag{3.19}$$

$$0 \leq t_{n,w,i} \leq q_{n,w+i} \text{ and } t_{n,w,i} \leq t_{n,w,i-1} \tag{3.20}$$

The complete weekend constraints $CW_n \in \{1, 0\}$ specify whether or not a nurse n should work either all days or no days at all during a weekend.[3] The total number of weighted constraint violations $V(CW)$ is defined in Expression (3.21).

$$V(CW) = \sum_{n \in N} W_n^{CW} \sum_{w \in W} CW_n(k.q_{n,w} - \sum_{i=0}^{k-1} p_{n,d_{w,1+i}}) \tag{3.21}$$

Similarly, the complete identical weekend constraints $CIW_n \in \{1, 0\}$ specify whether or not a nurse n should work the same shift type on all days in

[3] $CW_n = 1$ if a nurse n needs to work complete weekends, $CW_n = 0$ if she or he can work an arbitrary number of days in a weekend.

a complete weekend.[4] The total number of weighted constraint violations $V(CIW)$ is defined in Expression (3.22), subject to the inequalities in (3.23). The auxiliary variables $t_{n,w,s}$ in (3.23) indicate whether a nurse n is working shift type s during weekend w.

$$V(CIW) = \sum_{n \in N} W_n^{CIW} \sum_{w \in W} \sum_{s \in S} CIW_n (t_{n,w,s} k - \sum_{i=0}^{k-1} x_{n,d+i,s}) \tag{3.22}$$

$$-t_{n,w,s} k + \sum_{i=0}^{k-1} x_{n,d_{w,1+i},s} \leq 0 \text{ and } -t_{n,w,s} + \sum_{i=0}^{k-1} x_{n,d_{w,1+i},s} \geq 0 \tag{3.23}$$

The next set of constraints are related to individual requests. Nurses are allowed to request to be free or to be working on specific days or shifts. These requests are given by the respective request sets: D^{on}, D^{off}, S^{on}, and S^{off}. The total number of weighted constraint violations $V(D^{off})$, $V(D^{on})$, $V(S^{off})$ and $V(S^{on})$ is defined in the Expressions (3.24), (3.25), (3.26) and (3.27) respectively.

$$V(D^{off}) = \sum_{req_{n,d}^{off} \in D^{off}} W_n^{D^{off}} \sum_{s \in S} x_{n,d,s} \tag{3.24}$$

$$V(D^{on}) = \sum_{req_{n,d}^{on} \in D^{on}} W_n^{D^{on}} (1 - min\{1, (\sum_{s \in S} x_{n,d,s})\}) \tag{3.25}$$

$$V(S^{off}) = \sum_{req_{n,d,s}^{off} \in S^{off}} W_n^{S^{off}} . x_{n,d,s} \tag{3.26}$$

$$V(S^{on}) = \sum_{req_{n,d,s}^{on} \in S^{on}} W_n^{S^{on}} (1 - x_{n,d,s}) \tag{3.27}$$

The last set of constraints are called *global constraints*, meaning that they are set only once and apply for all nurses together. Consequently, there is no individual weight associated to these constraints. Instead, one collective weight can be set.

[4] $CIW_n = 1$ if a nurse n should work identical days during a weekend, $CIW_n = 0$ if he or she can work different shift types during the days of a weekend.

The alternative skill constraint $SK^{\text{alt}} \in \{0, 1\}$ specifies whether or not nurses can be assigned shift types for which they do not have all the required skill types.[5] The weighted number of constraint violations $V(SK)$ is defined in Expression (3.28), subject to the inequalities in (3.29). The auxiliary variables $r_{n,s}$ in Expression (3.29) denote whether a nurse n has the required skill types to work shift s. The variables $b_{n,sk}$ denote whether nurse n has skill type sk.[6] Hence, the variables $t_{n,d,s}$ in (3.28) equal 1 if and only if nurse n works shift type s on day d, while not having all the required skill types.

$$V(SK) = SK^{\text{alt}}.W^{SK} \sum_{n \in N} \sum_{d \in D} \sum_{s \in S} t_{n,d,s} \tag{3.28}$$

$$\begin{aligned} & t_{n,d,s} \geq x_{n,d,s} - r_{n,s} \\ \text{and } & -r_{n,s}|SK_s| + \sum_{sk \in SK_s} b_{n,sk} \geq 0 \\ \text{and } & -r_{n,s} - |SK_s| + 1 + \sum_{sk \in SK_s} b_{n,sk} \geq 0 \end{aligned} \tag{3.29}$$

Certain patterns in the schedule of a nurse should always be avoided. In this dissertation, we consider two types of patterns in the problem description. The first pattern type specifies a series of consecutive shifts (starting on any day) which should be avoided. Each nurse n has a set PAT_n^{cons} of unwanted patterns *pat*. Each pattern *pat* is a sequence of shift types s_i and has a certain length, denoted by $length_{pat}$. The second pattern type avoids nurses to have a free day before working one of the successive days.[7] Similarly, each nurse n has a set PAT_n^{free} of unwanted patterns *pat*. Each pattern *pat* now consists of a starting day d_{pat} (i.e. the day that should be free) and a certain length $length_{pat}$. The total number of weighted violations $V(PAT^{cons})$ and $V(PAT^{free})$ are defined in Expressions (3.30) and (3.31) respectively.

The main advantage of a mathematical model like the one presented here, is its clear and exact formulation. This is especially important for the evaluation of the constraints. As Haspeslagh (2012) has noted, such an unambiguous

[5] $SK^{\text{alt}} = 0$ if nurses are allowed to work shift types for which they do not have all the required skills, $SK^{\text{alt}} = 1$ otherwise.

[6] $b_{n,sk} = 1$ if $sk \in SK_n$ and $b_{n,sk} = 0$ otherwise.

[7] For example, a pattern could state that a nurse should not be free on the Friday preceding a working weekend.

evaluation scheme is not always present, leading to different interpretations of how certain constraint violations should be counted in the objective function.

$$V(PAT^{cons}) = \sum_{n \in N} W_n^{PAT^{cons}} \sum_{d \in D} \sum_{pat \in PAT_n^{cons}} max\{((\sum_{s_i \in pat} x_{n,d+i,s_i}) - length_{pat} + 1), 0\} \tag{3.30}$$

$$V(PAT^{free}) = \sum_{n \in N} W_n^{PAT^{free}} \sum_{pat \in PAT_n^{free}} max\{(min\{\sum_{i=1}^{lentgh_{pat}} p_{n,d_{pat}+i}, 1\} - (1 - p_{n,d_{pat}})), 0\} \tag{3.31}$$

3.3 Performance prediction for nurse rostering

In this section, we will focus on the large experimental study we have performed on nurse rostering problems. The main aim of this study is to build accurate performance prediction models for a set of practical nurse rostering instances. The study can be divided into two main parts. First, as a proof of concept, we focus on small, relatively simple nurse rostering problems. The goal here is to investigate whether it is altogether possible to build performance prediction models for nurse rostering. By keeping the instance size small, we have the advantage that we can look at both complete search methods and metaheuristic techniques. This first part of the experimental study is described in Section 3.3.1. Second, in Section 3.3.2, we look at larger nurse rostering instances based on experience with real-world data. This second part focuses on practical nurse rostering problems. Due to the size and the complexity of such problems, it is no longer possible to apply complete search methods. The running times would simply be too long. Instead, we focus on two state-of-the-art metaheuristics that competed in the 2010 International Nurse Rostering Competition (INRC 2010).

Both sections will follow the structure of the five-step generalised framework for building algorithm performance prediction models as proposed in Section 2.2.

3.3.1 Proof-of-concept study

Step 1: Instance distribution

The first step in the strategy sets the scope of the research. The problem instance distribution under study can be categorised as $SA|V3|P$ in the classification of De Causmaecker and Vanden Berghe (2011). We consider sequence constraints (S) and availabilities (A) with a fluctuating coverage (V) in a 3-shift structure. The planning horizon for these problems is two weeks. The three shift types are labelled: `Early`, `Late` and `Night`. In total, there are thus 42 distinct time frames that need to be covered. For these instances, we do not distinguish between different skill types. All nurses have only one skill type and all shift types require exactly this skill type. The coverage constraints are fixed and equal for all instances. There is a nurse pool consisting of six identical nurses working according to one and the same contract type. This contract specifies the sequence constraints on the individual work schedules. The constraint values of the contract are randomly drawn from appropriate intervals. In this limited study, we do not consider unwanted patterns or special requests. Nurses are required to work complete and identical weekends and can not work more than one shift type per day. The soft constraints are not weighted (i.e. the weight associated to all constraints equals 1).

This instance distribution is deliberately kept simple. The aim of the experiment is to provide a proof of concept. By keeping the instances small and relatively uncomplicated, we can investigate their empirical hardness for both complete and incomplete search methods. By varying only the contract constraints, we focus on the influence of these constraints on the empirical hardness, while eliminating the effect of size parameters (like e.g. the number of nurses, days, shifts, etc.).

Step 2: Algorithm set

As mentioned above, the size of the considered instances is kept small, allowing us to investigate whether it is possible to build performance prediction models for both complete and incomplete search methods. One of the reasons for this is the analogy with the work of Leyton-Brown et al. (2006) and Nudelman et al. (2004), where the running time of a complete search method is the target value to be predicted.

The first algorithm we consider is the commercial IBM ILOG CPLEX optimiser. This solver is run on an integer program representation (based on the mathematical model of Section 3.2.1) of the instances and is guaranteed to find optimal solutions, when given sufficient calculation time. Apart from the running time, we also record the *quality* of the optimal solution as a measure for the empirical hardness of an instance. This quality is measured as the number of constraint violations. The lower this value, the higher the solution quality.

The second algorithm in this study is a metaheuristic local-search method. It is a variable neighbourhood search based on the neighbourhoods described by Burke et al. (2008). Metaheuristics are in general unable to prove optimality for the produced solutions, or to give upper or lower bounds on the optimality gap.[8] Typically, metaheuristics are allowed to search for a solution for a predetermined amount of time. This time is usually expressed as an effective running time, although sometimes other stopping criteria can be used, such as e.g. a fixed number of objective function evaluations. For this experiment, we let the metaheuristic run for a fixed amount of time (4 minutes), based on the observations of the original authors. The performance of the metaheuristic is measured as the quality of the produced solutions. The average value over five independent runs is recorded.

In the context of this experiment, we can look at an additional performance measure for the metaheuristic algorithm. Since the quality of the optimal solutions is known (as a result of running the complete search method), we can also consider the quality gap of the metaheuristic solutions.

Step 3: Feature selection

The prediction of performance criteria through the use of so-called empirical hardness models requires the selection of a feature set adequately characterising the instances as vectors of feature values. For such models to be effective in making quick predictions, these feature values should be efficiently computable.

In this experiment, we will investigate several approaches. We will design a feature set specifically for this instance distribution, characterising the instances by means of several properties and constraint values. Apart from this domain-specific approach, we will look at a feature set which was originally designed for a much more general problem domain. More specifically, we will investigate the applicability of the feature set for SAT problems (Nudelman et al., 2004), described in Section 2.2.1.

[8] The optimality gap is the difference in quality between the found solution and an optimal solution.

As we discussed in Section 2.2.1, designing a domain-specific feature set is not straightforward. However, given the limitations of the problem instance distribution considered here, a number of natural feature candidates can easily be found. Only the contract constraint values are varied over the problem instances. These values are furthermore the same for all nurses in the pool. Evidently, these constraint values can be included as features in the set. Additionally, we included a number of features related to structural properties of the instances. Reflecting on the problem domain (given the experience of our research team with nurse rostering problems) revealed a number of properties which could maybe be linked to the empirical hardness of the instances. These features can be interpreted as indicators of the tightness of the constraints, or in other words, as indicators of the degree of freedom in solving the instances. For example, we include a feature (the `ratioAvailabilityOverCoverage`) representing the total number of shifts that can maximally be worked by the nurses (based on their maximal number of assignments) divided by the number of shifts that need to be worked (based on the coverage requirements). If this ratio equals 1, then all nurses should work their maximum allowed number of assignments to be able to cover all shifts. The higher this ratio, the more freedom there is in distributing the work over the nurses. The NRP feature set is thus based on an intuitive understanding of the problem domain and consists of the following 11 features:

- *features related to the constraint values defined in the contract:* (6)
  - maximum and minimum total number of assignments
  - maximum and minimum number of consecutive working days
  - maximum and minimum number of consecutive free days
- *features related to structural properties:* (5)
  - indicating the hardness of feasibility:
    - the maximum number of shifts that can be covered by all nurses divided by the total number of shifts that need to be covered (denoted as `ratioAvailabilityOverCoverage`)
    - the maximum number of assignments divided by the minimum number of consecutive working days
  - indicating the tightness of constraints:
    the maximum-over-minimum ratio of
    - the number of assignments
    - the number of consecutive working days
    - the number of consecutive free days

As an alternative to the NRP feature set, we selected a subset of features from the existing SAT feature set of Nudelman et al. (2004). Before such features can be calculated, the nurse rostering instances must first be transformed into SAT instances. For this purpose, we employed the efficient translation scheme of Haspeslagh et al. (2013) (which we co-authored). It is important to note that SAT problems are decision problems, which contrasts to the optimisation aspect of nurse rostering. The translation scheme thus treats all constraints as hard constraints. Consequently, solving the resulting instances by means of a SAT solver quickly leads to the decision that no model can be found satisfying all boolean constraints. Having this in mind, it is thus not at all clear whether the SAT features will be informative for the empirical hardness of nurse rostering problems (or more generally the empirical hardness of optimisation problems). We have selected a subset of 50 SAT features from the following groups and refer to Section 2.2.1 for more details on these features:

- *features related to the problem size* (11)
- *features based on the problem structure* (20)
- *features corresponding to the balance of the formula* (13)
- *features measuring the proximity of an instance to a Horn formula* (6)

We have selected only domain-dependent features which are easily computable by simply iterating over the clauses and variables of the SAT problem instances. The remaining features in the original description of the feature set require more computational effort and are left out for that reason.

There exist other possibilities too. One could e.g. generate a feature set based on the integer program description of Section 3.2.1. This is however outside the scope of this dissertation. Hutter et al. (2014) give an overview of methods aiming at algorithm runtime predictions. They present a feature set for MIP formulations. Many of these features however, are covered by our NRP feature set (based on the constraint values and their interactions).

Step 4: Data generation

As briefly mentioned in Chapter 2, the acquisition of qualitative training data is important for the success of a any model based on this data. We have built a random instance generator producing nurse rostering instances belonging to the instance distribution defined above. This generator simply chooses random constraint values from the intervals defined in the instance

distribution. Two separate datasets are generated: a smaller one and a larger one. The smaller dataset contains 600 random instances, the larger one 3000 (including those instances of the smaller dataset). As discussed in Section 2.2.2, it is recommended to have separate datasets for training and evaluating the prediction models. Therefore, we randomly partitioned both datasets into a training set and a validation set. The smaller dataset is partitioned into a training set of 500 instances and a validation set of 100 instances. The larger one is partitioned into a training set of 2000 instances and a validation set of 1000 instances.

The performance criteria are determined for all generated instances. For the smaller dataset, both the complete solver and the metaheuristic algorithm are run. Using the complete solver, the quality of the optimal solution and the running time needed to find this optimal solution is determined. Additionally, we also compute the logarithm of the running time. Using the metaheuristic algorithm, the quality of the solutions after 4 minutes of running time is determined. Based on this information, we also compute the quality gap. This quality gap provides an indication of the relative quality of the metaheuristic algorithm, given its stopping criterion. On the larger dataset, we apply only the metaheuristic algorithm. As a consequence, the only information available is the quality of the metaheuristic solution. Although the instances are sufficiently small for the complete search method to be feasible, running this solver on 3000 instances is still rather impractical. Nevertheless, the smaller dataset is sufficiently large to produce meaningful results. The larger dataset should thus only be seen as an additional dataset, providing more information on the performance of the metaheuristic algorithm.

Determining the performance measures for all problem instances is only half of the required work in this data generation step. The second part is calculating the feature values for all problem instances. The NRP feature values can easily be determined. For the calculation of the SAT feature values, a translation to SAT problems is carried out. As mentioned in Section 2.2.2, a data preparation phase is recommended before any machine learning technique is applied. Given the limited variation in this proof-of-concept instance distribution, it is expected that some of the features will be superfluous or even useless. Features that are uni-valued, meaning that their value is constant throughout the dataset, are eliminated. Furthermore, a correlation analysis is carried out and perfectly correlated features are removed. The NRP feature set did not contain uni-valued features. This is not really surprising, the feature set was specifically designed for this limited instance distribution, and any features that would be uni-valued (e.g. size related features) were not included to begin with. For the same reason, there are no perfectly correlated features in the NRP feature set. The SAT feature set on the other hand did contain a certain amount of uni-valued or

perfectly correlated features. These features were easily filtered out and the remaining set contains the following 28 features:

- *features related to the problem size:* (1)
  - clauses-to-variables ratio
- *features based on the problem structure:* (14)
  - features based on the variable-clause graph:
    - clause node degree: mean, variation, standard deviation and maximum over all clause nodes
    - variable node degree: mean, variation, standard deviation and minimum over all variable nodes
  - features based on the clause graph:
    - node degree: mean, variation and standard deviation over all nodes
  - features based on the variable graph:
    - node degree: mean, variation and standard deviation over all nodes
- *features corresponding to the balance of the formula:* (9)
  - the ratio of positive literals per clause: mean, variation and standard deviation over all clauses
  - the ratio of positive occurrences per variable: mean, variation and standard deviation over all variables
  - the portion of unary, binary and ternary clauses in the SAT formulation
- *features concerning the proximity to Horn formulae:* (4)
  - the fraction of Horn clauses in the formula
  - the number of occurrences of a variable in a Horn clause: mean, variation and standard deviation over all variables

Step 5: Building performance prediction models

The main objective of this experiment is to investigate whether it is feasible to build accurate performance prediction models for nurse rostering problems. All necessary ingredients are now available. There is a sufficient amount of

training data for which we have information on the performance of two different algorithms. All problem instances have furthermore been characterised by two distinct feature sets: one specific to the nurse rostering domain, the other one based on a translation to SAT problems. Remaining to be done is the actual model construction for the different performance criteria relevant in this setting. There are numerous possibilities for building such models. Inspired by the studies of Leyton-Brown et al. (2006) and Nudelman et al. (2004), we start with linear regression techniques. Later on, we will also apply other machine learning techniques.

For the different performance criteria (running time of the complete search method, quality of the optimal solution, quality of the metaheuristic solution and the quality gap), we build and compare three types of models. The first are based on NRP features, the second are based on SAT features, and the last are based on the combination of both feature sets. All models are built using only the training instances and evaluated using 10-fold cross-validation. The use of cross-validation provides a more realistic view on the performance of the model on unseen data. The best models are finally evaluated on the validation set of unseen instances, in order to determine their true performance on instances outside the training set.

The results of these experiments are discussed for each performance criterion separately in the following subsections.

Running time of the complete search method

When examining the running times of the complete solver on the training instances, we observe very large fluctuations. Of the 500 instances, 370 were solved in less than 20 milliseconds. For a much smaller number of instances, the solver needed several orders of magnitude more time (up to more than three hours of running time). Due to these high fluctuations, it can be expected that applying a purely linear regression model will not be the best choice. Indeed, the correlation coefficients on the training set (using 10-fold cross-validation) are lower than $R = 0.30$, regardless of the feature set being used. Low correlation coefficients indicate poor predictive behaviour.

One approach that might lead to more accurate results is through a logarithmic transformation of the running times. Such a transformation largely reduces the fluctuations in the dataset. In the studies of Leyton-Brown et al. (2006), Nudelman et al. (2004) and Xu et al. (2008), the empirical hardness models for the logarithm of the running time are very accurate. However, in our experiment, the linear regression models for the logarithm of the running time perform only marginally better than those for the plain running time. The

correlation coefficients on the training set are still not higher than $R = 0.35$ (using 10-fold cross-validation).

There is no clear explanation for this finding. While linear regression techniques appear to be well-performing in other domains, this is definitely not the case for our small nurse rostering problems, regardless of the feature set being used. One possible cause for this could maybe be found in the integer programming formulation which has not been optimised. Certain expressions of the constraints could perhaps lead to a huge increase in running time on certain instances. Note that these factors actually fit into the framework of empirical hardness models (i.e. it is the combination of an instance (representation) and an algorithm that determines the performance); but the linear regression techniques seem to be inadequate for predicting the behaviour of the complete solver. Other mathematical formulations might be better suited for the solver. The feature sets are based on the original formulation, and not on the mathematical program. There is no information on the mathematical program present in the features, and hence it is hard to find an accurate model predicting (a function of) the running time based on these features. There is however no empirical evidence for this statement (yet). Bearing in mind the goals and limitations of this proof-of-concept study, we did not think it to be opportune to pursue an investigation of whether a better formulation would lead to more accurate results, or whether a feature set based on the mathematical program would make a difference. Hardly any application of such ideas to practical nurse rostering instances (or maybe even none at all) will be considering complete search methods.

Other machine learning techniques than linear regression are also applicable. We have applied an extensive set of techniques, made available through the Weka software tool (Hall et al., 2009). We have tried to build models predicting the plain running time and the logarithm of this running time. With respect to the plain running time, none of the considered techniques achieved better correlation coefficients than the linear regression models. For the logarithm of the running time on the other hand, certain techniques were able to clearly produce better results. This is probably due to the smaller fluctuations in the logarithm of the running time. Table 3.2 summarises the results of a number of other techniques in Weka. It presents the correlation coefficients of various models predicting the logarithm of the running time, evaluated using 10-fold cross-validation on the training set. Apart from linear regression, it includes rule sets (Decision Table and M5 Rules) and regression trees (M5P Tree). Rule sets are based on a number of if-then-else rules and employ one of a series of regression models depending on the rules that apply. Regression trees are in fact decision trees with regression models at the level of the leaves. The three columns in the table correspond to the feature sets on which the models are based. For all considered models, it appears that using the NRP feature

	NRP features	SAT features	NRP + SAT features
Linear Regression	0.2528	0.2583	0.2364
Decision Table	0.9118	0.3911	**0.9118**
M5 Rules	0.9052	0.4145	0.9044
M5P Tree	**0.9135**	**0.5290**	0.8879

Table 3.2: Correlation coefficients (R) of various models predicting the logarithm of the running time of the complete search method, based on 10-fold cross-validation on the training set.

set alone allows for better predictions than using the SAT feature set alone ($R = 0.91$ versus $R = 0.53$). This could however be due to the limitations of the problem instance distribution. Only few of the parameters are varied, and these parameter values are all directly included in the NRP feature set. The translation to SAT results in a much more aggregated view on these parameters, expressed in a set of SAT features. Feeding these SAT features to a machine learning technique might complicate the process of building an accurate model.

Although the results of other machine learning techniques are significantly better than those of the linear regression models, these other techniques are still not producing highly accurate models. Figure 3.1 presents a plot of the actual versus the predicted logarithm of the running time on the training instances, using the M5P tree model based on NRP features (having a correlation coefficient of $R = 0.91$). At first sight, it could be thought that this graph is incorrect, as the predictions are generally too small for large running times and too large for small running times. However, the reason for this is an extremely high density of data points in the lower left section of the graph (which is not really visible in Figure 3.1). Applying an additional linear regression to these results does not further improve on the accuracy. Anyhow, the prediction errors of this model are still rather high. Given this instance distribution, it appears to be hard to accurately predict the (logarithm of the) running time.

There are however other properties that might be more predictable. Instead of predicting the actual running time, it might be useful to predict whether or not this time will be *acceptable*, regardless of how long it is exactly. We

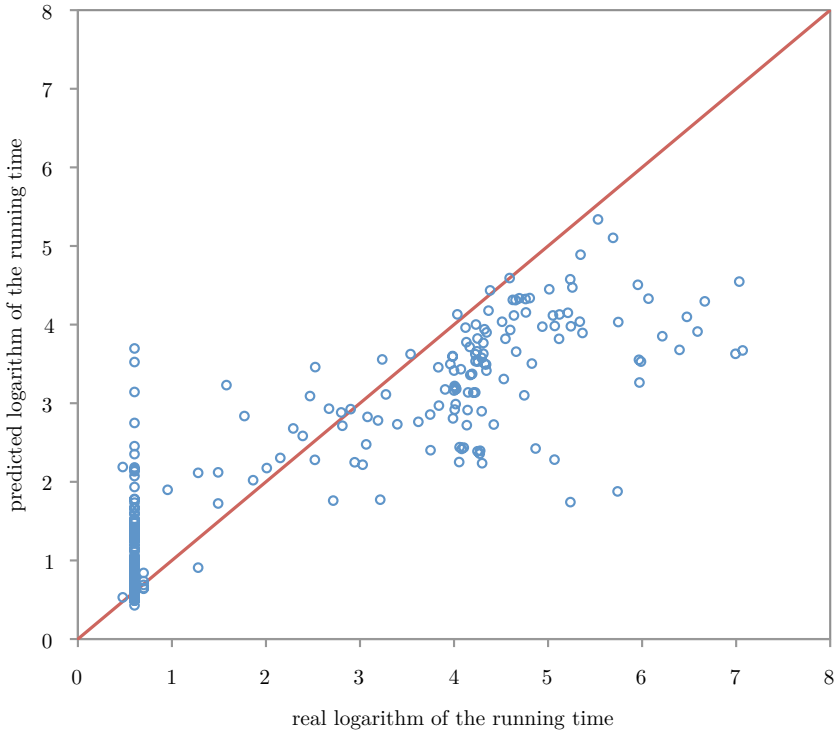


Figure 3.1: Actual versus predicted logarithm of the running time of the complete search method on the training set. Predictions of the M5P tree model based on the NRP feature set.

have therefore extended the training set to include a class attribute indicating whether the running time of the complete search method is more or less than 20 milliseconds. Note that this particular threshold is arbitrary and chosen based on an inspection of the training set. In a practical situation, this threshold could be set according to other criteria (e.g. a clustering approach resulting in two or more classes). We applied various classification techniques building models predicting whether the instance will be solved quickly or not. Table 3.3 presents the results of this approach. It shows the percentage of correctly classified instances for a subset of the methods we have employed, evaluated using 10-fold cross-validation on the training set. It includes decision trees (J48), random forests, decision tables and naive Bayes classifiers. As can be seen from the table, the success rate of several methods is very high. The best model is based on random forests using only NRP features. This is a classification

	NRP features	SAT features	NRP + SAT features
J48	95.8%	**86.2%**	**95.6%**
Random Forest	**96.8%**	74.0%	94.8%
Decision Table	94.8%	75.9%	94.2%
Naive Bayes	87.8%	66.4%	70.2%

Table 3.3: Percentage of correctly classified instances of various models predicting the feasibility of a complete search method, based on 10-fold cross-validation on the training set.

technique which builds a collection of (in our case 500) decision trees for which the features are randomly selected. The final prediction is based on a majority vote among the decision trees. The models based on NRP features generally perform better than those based only on SAT features. Combining both sets does not improve the results. In this context, it appears that the SAT feature set is not well-suited for predicting running time-related properties. As suggested before, this could be related to the translation to SAT problems, complicating the learner's view on the instances.

Quality of the optimal solution

The second criterion of interest is the quality of the optimal solutions, measured by a cost function counting the number of constraint violations. This is strictly speaking not really an algorithm performance measure. It is actually a property of the instance which is not affected by the choice of the algorithm. Nevertheless, it is an interesting property in itself, but also in relation to the solution quality of an incomplete method (which will be discussed further on). Quality predictions have a number of practical applications, which will be discussed in Section 3.4.

Using various techniques in Weka, we built a number of prediction models for the quality of the optimal solution. The results are shown in Table 3.4, which gives an overview of the correlation coefficients of various models, evaluated using 10-fold cross-validation on the training set. The correlation coefficients are very high, indicating very good predictive power. Indeed, when the best model

	NRP features	SAT features	NRP + SAT features
Linear Regression	0.9694	**0.9678**	0.9744
M5P Tree	0.9823	0.9666	**0.9821**
Decision Table	0.9555	0.7513	0.8909
M5 Rules	**0.9843**	0.9666	0.9813

Table 3.4: Correlation coefficients (R) of various models predicting the quality of the optimal solution, based on 10-fold cross-validation on the training set.

(the M5 Rules model based on NRP features) is evaluated on the validation set of unseen instances, a similar correlation coefficient of $R = 0.9719$ is achieved. Figure 3.2 shows a plot of the actual versus the predicted optimal quality by this model, evaluated on the validation set of unseen instances. It indicates that the predictions are indeed strongly correlated to the actual optimal qualities.

In contrast to the results for predicting (a function of) the running time, we find no significant difference in performance between the best models based on the NRP feature set and those based on the SAT feature set. Both feature sets allow for accurate predictions of the quality of the optimal solution. Combining both feature sets does not further improve the results. This observation provides a stronger motivation for our previous statement that the integer programming formulation could have had an influence on the results for predicting the running time of the complete solver. The high fluctuations in running times might indeed be an artefact of the mathematical formulation, for which no information is present in the feature sets. This could lead to a generally lower quality of the prediction models for the (logarithm of the) running time. Looking at the results for predicting the quality of the optimal solution, the (poor?) mathematical formulation has no effect and the results are significantly better.

Quality of the metaheuristic solution

In contrast to complete search methods, the quality of the solutions of incomplete methods is a clear algorithm performance measure. Such methods in general, and metaheuristics in particular, produce fairly good solutions in a reasonable

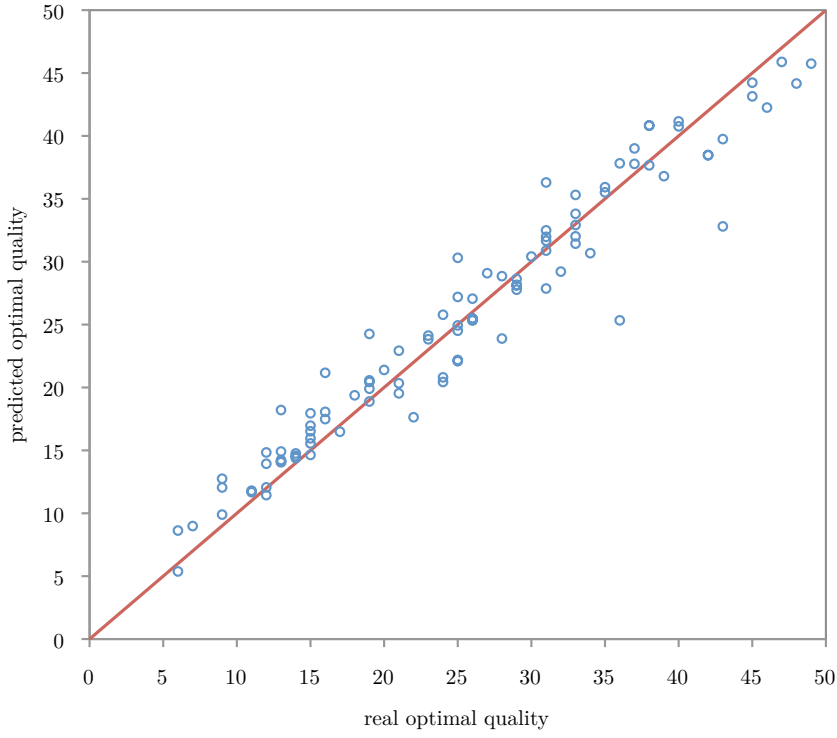


Figure 3.2: Actual versus predicted quality of the optimal solution on the validation set. Predictions of the M5Rules model based on the NRP feature set.

amount of time. These solutions could be optimal, but there is generally no way of knowing whether this is the case. For this performance criterion, there is no information needed on the complete solver. As a consequence, we can use the larger dataset to construct and evaluate models for the prediction of the solution quality.

Table 3.5 summarises the results of building performance prediction models for the quality of the metaheuristic solutions. It shows the correlation coefficients for various models, evaluated using 10-fold cross-validation on the training set. We find similar results as for the predictions of the quality of the optimal solutions. The most accurate models are again those based on the NRP feature set. We see that both feature sets allow for accurate performance predictions. The correlation coefficients of the models based on NRP features are slightly higher than those of the models based on SAT features. Combining both

	NRP features	SAT features	NRP + SAT features
Linear Regression	0.9699	0.9587	0.9773
M5P Tree	0.9936	0.9741	**0.9899**
Decision Table	**0.9952**	0.8321	0.9552
M5 Rules	0.9931	**0.9753**	0.9883

Table 3.5: Correlation coefficients (R) of various models predicting the quality of the metaheuristic solution, based on 10-fold cross-validation on the training set.

feature sets does not lead to more accurate results. We further evaluated the Decision Table model based on NRP features on the validation set of 1000 unseen instances. The resulting correlation coefficient is $R = 0.9965$. Figure 3.3 shows a plot of the actual versus the predicted metaheuristic quality by this model, for the instances in the validation set. We can see that the predictions are indeed very accurate. The results are even slightly better than for predicting the optimal quality of the solutions, as the data points lie even closer to the diagonal. With respect to the goals of this proof-of-concept study, these are very important results. We have empirically proven here that performance predictions for practical combinatorial optimisation problems like the nurse rostering problem are possible. The results even show that these predictions are very accurate, achieving very high correlation coefficients. Part of the reason why these correlation coefficients are this high, could be related to the simplicity of the instance distribution. Only few, but important, parameters are varied, leading to a relatively homogeneous set of instances which could facilitate the predictions. This is however only speculative. When examining more realistic instance distributions later on, we will eliminate this effect and will see that similar correlation coefficients can be achieved there.

Optimality gap

Given the use of a complete solver in this proof-of-concept study, information is available on the optimal solutions for the instances in the smaller dataset.

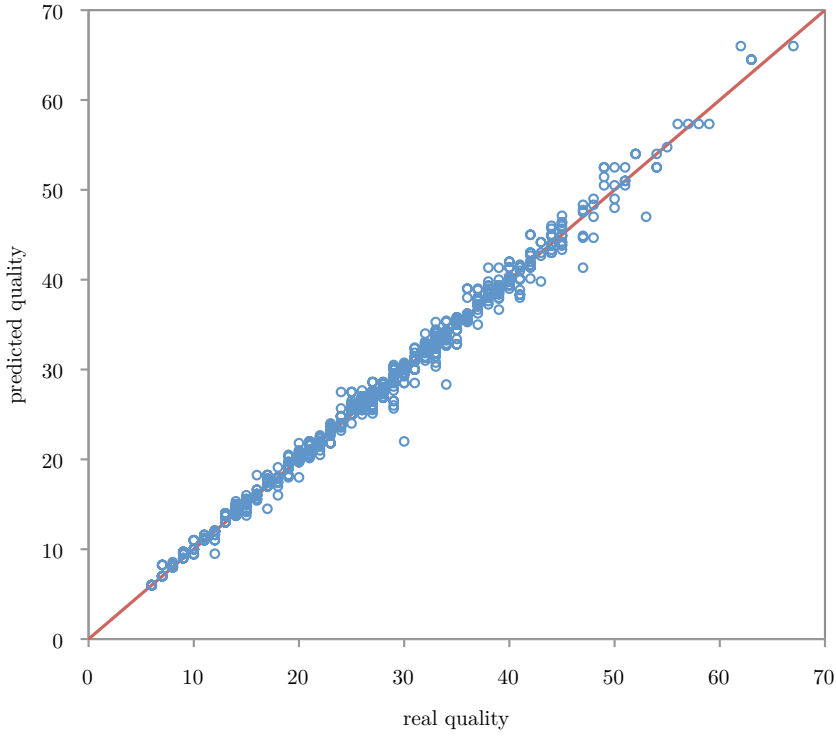


Figure 3.3: Actual versus predicted metaheuristic quality on the validation set. Predictions of the Decision Table model based on the NRP feature set.

This information can be used to assess the performance of the metaheuristic in an additional way. By subtracting the quality of the optimal solution from that of the metaheuristic solution, we find a distance indicating how far the metaheuristic solution is removed from the optimal solution.[9] We can either consider this absolute gap, or we can look at the relative gap with respect to the quality of the optimal solution.

The correlation coefficients of various models, evaluated using 10-fold cross-validation on the training set, are shown in Tables 3.6 and 3.7, for the prediction of the absolute and the relative gap respectively. The correlation coefficients are not very high, indicating poor predictive power of the models. Indeed, when the M5P Tree model, based on the combination of NRP and SAT features is evaluated on the validation set of unseen instances, the correlation coefficient

[9]Remember that qualities are measured by the number of (weighted) constraint violations.

	NRP features	SAT features	NRP + SAT features
Linear Regression	0.6248	0.6344	0.6610
M5P Tree	0.6828	0.6765	**0.6966**
Decision Table	**0.7007**	0.6413	0.6868
M5 Rules	0.6949	**0.6855**	0.6819

Table 3.6: Correlation coefficients (R) of various models predicting the absolute quality gap, based on 10-fold cross-validation on the training set.

	NRP features	SAT features	NRP + SAT features
Linear Regression	0.5617	0.5769	0.5980
M5P Tree	0.5716	0.5973	**0.6226**
Decision Table	**0.6018**	0.5203	0.5725
M5 Rules	0.5711	**0.5915**	0.6177

Table 3.7: Correlation coefficients (R) of various models predicting the relative quality gap, based on 10-fold cross-validation on the training set.

	NRP features	SAT features	NRP + SAT features
baseline	63.2%	63.2%	63.2%
J48	**86.8%**	80.2%	**83.2%**
Random Forest	83.6%	**81.4%**	81.6%
Decision Table	82.2%	78.4%	78.8%
Naive Bayes	81.2%	71.6%	72.8%
Multilayer Perceptron	78.2%	79.8%	81.2%

Table 3.8: Percentage of correctly classified instances of various models predicting the feasibility of a complete search method, based on 10-fold cross-validation on the training set.

is only $R = 0.4139$. Figure 3.4 shows a plot of the actual versus the predicted relative quality gap by this model, evaluated on the validation set. We see that the data points are indeed not close to the diagonal.

It might seem strange at first, that both the optimal quality and the metaheuristic quality can be fairly accurately predicted, but not the (relative) difference of both qualities. However, for 63% of the instances, the quality gap is 0%, meaning that the metaheuristic was able to find an optimal solution. In only 10% of the instances, the relative quality gap is larger than 10%. In absolute numbers, the difference is larger than 3 for only 5% of the instances. Bearing in mind that the optimal quality varies from 6 to 67, we can state that the metaheuristic is performing very well on these instances, achieving optimal results in many cases. Even when it is not reaching an optimal solution, the result is never far from optimal. There is no straightforward explanation to why the metaheuristic is not finding optimal solutions for these instances. The algorithm might get stuck in very wide areas around certain local optima. We did however not pursue further investigation to why this is the case. The main focus of this experiment is to demonstrate the feasibility of performance predictions in this limited setting.

Looking at this dataset where the metaheuristic is solving many instances to optimality, it could be interesting to know in advance whether this is going to be the case or not. We have therefore added a classification attribute to the data,

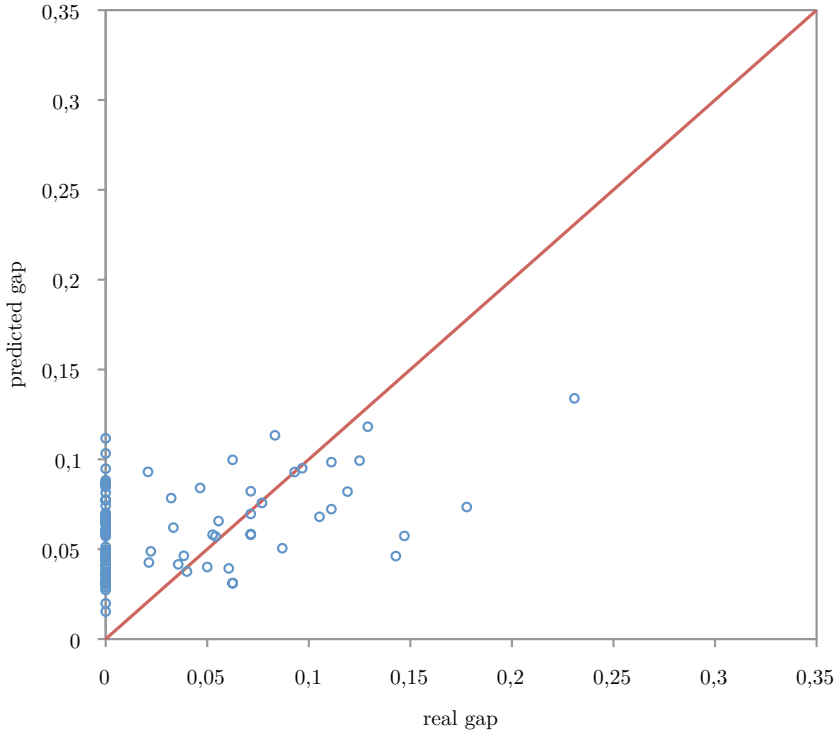


Figure 3.4: Actual versus predicted relative quality gap on the validation set. Predictions of the M5P Tree model based on the combination of the NRP and SAT feature set.

indicating the binary decision of whether the metaheuristic was able to find an optimal solution or not. We then applied various classification techniques in order to build prediction models for this decision. The results are summarised in Table 3.8. It presents the percentage of correctly classified instances of various methods, based on 10-fold cross-validation on the training set. Note that the baseline to compare with is a classifier that always predicts a `yes` answer. The percentage of correctly classified instances of such a classifier is 63.2%. The best models for this objective are based on decision tree learners. Their performance is significantly better than the baseline, but still not highly accurate, achieving percentages around $80\% - 85\%$.

3.3.2 Performance prediction for challenging instances

Now that we have established the feasibility of building accurate performance prediction models for basic nurse rostering instances, we will apply these ideas to more interesting problem instances. In contrast to the proof-of-concept experiment, we will here include all bells and whistles of the nurse rostering model. We will thus be considering practical problem instances with real-world relevance.

Step 1: Instance distribution

The instance distribution is based on the medium track of the 2010 International Nurse Rostering Competition.[10] This competition is in its turn inspired by real-world applications and problem instances. We look at instances of similar size, with similar nurse characteristics and similar coverage requirements. The problem instance distribution can be described by $SA2|V5|P$ in the classification of De Causmaecker and Vanden Berghe (2011). We consider different sequence constraints (S) and availabilities (A) with a fluctuating coverage (V) in a 5-shift structure. The shift types in the problem are labelled `Day`, `Early`, `Late`, `Night` and `DayHead`. The `DayHead` shift type requires a nurse that owns two skill types, the regular 'nurse' skill type and the 'head nurse' skill type. The other shift types require nurses that own the regular 'nurse' skill type. These skill types are the only skill types present in the problem formulation. The planning horizon of the instance distribution is either two or four weeks (with uniform probability). The coverage constraints are randomly chosen per week and differ for week and weekend days. The required number of nurses per shift type are random values in appropriate intervals. The number of nurses in the nurse pool is varied between 20 and 40, both inclusive. In this distribution, all nurses have the standard 'nurse' skill type. Four of these nurses have the additional 'head nurse' skill type. Each nurse works according to one out of four prototypical contracts. Each contract specifies all the sequence constraints on the personal schedules of the nurses. The different contracts represent whether nurses work full-time, part-time, 75% or 80%. All head nurses are full-time employed. The constraint values in the contracts are randomly chosen from appropriate intervals. Additionally, each nurse has a random number of individual `DayOn`, `DayOff`, `ShiftOn` and `ShiftOff` requests. The soft constraints are unweighted (i.e. their weight is set to 1). All random values are drawn from uniform distributions over appropriate intervals based on the medium track of the 2010 INRC.

[10] This competition had three distinct tracks: sprint, medium and long, corresponding to the maximum allowed calculation time. In the medium track, the algorithms were allowed to run for 10 minutes.

Step 2: Algorithm set

Due to the size and the intrinsic complexity of the instance distribution, we can not employ complete search methods like we did in the proof-of-concept study. Moreover, the variable neighbourhood search metaheuristic used in that study is too simple and not sufficiently powerful for these larger nurse rostering instances. The scope of this experiment is thus very different from the proof-of-concept study. We are no longer looking at artificial, small instances. Instead, we are interested in more practical instances with real-world relevance. As a consequence, we have to look at qualitative algorithms able to tackle these large, practical nurse rostering instances. The literature provides us with a number of choices, but as we have argued before, many algorithms are specifically tailored to the problem formulation of the respective research paper. Because of the instance distribution we consider, it is natural to look at algorithms that competed in the 2010 INRC. For practical reasons, we have based our algorithm choice on the availability of a UNIX-executable solver. As a result, we consider two of the top-five competitors in the competition.

The first algorithm, denoted as `Algorithm A`, is the hyper-heuristic developed by Bilgin et al. (2010). In this hybrid approach, a state-of-the-art hyper-heuristic (based on the work of Özcan et al., 2008) is deployed in the first 80% of the time, and a much older greedy shuffle heuristic (Burke et al., 1999) is applied during the remainder of the time, improving the solution of the hyper-heuristic.

The second algorithm is based on a tabu search algorithm for general constraint optimisation problems and is developed by Nonobe (2010). The algorithm uses a fairly simple neighbourhood and dynamically controls the tabu tenure based on intermediate results.

As the performance of both algorithms during the competition was very good, we can consider them to be state of the art. Please note that the choice for these two particular algorithms is arbitrary. Other choices are possible, but we have selected these for the reasons mentioned above.

The performance of both algorithms will be assessed by the quality of the solutions they produce. To allow for a fair comparison, both algorithms are given the equivalent of 10 minutes of calculation time on a prototypical machine. This stopping criterion is the same one as used in the medium track of the 2010 INRC. The quality of the solutions is measured by the sum of weighted constraint violations, similar to the proof-of-concept experiment. The lower this value, the better the solution quality. In order to minimise the impact of random decisions in the solution process, the average performance of the algorithms over five independent runs will be considered.

Step 3: Feature selection

In order to build empirical hardness models for the prediction of algorithm performance, we must be able to efficiently characterise the problem instances by a vector of feature values. The NRP feature set introduced in Section 3.3.1 is not sufficiently detailed to capture all the properties of the instances in this larger experiment. Indeed, the feature set was only considering the properties of the instances which were actually varied over the instance distribution. For example, all problem instances considered the same number of nurses. Consequently, this property was not considered as a feature. It would have been uni-valued and thus deleted anyhow. Additionally, in the proof-of-concept distribution, all nurses are working according to the same contract. All nurses thus had the same constraint values on their personal schedules. This is no longer the case for the instances considered in this larger study. As a consequence, we need to design a more elaborate feature set. Instead of the 11 features being used in the proof-of-concept study, we have built an extensive set containing 305 features specific to the nurse rostering formulation considered in this dissertation. We describe this feature set in detail in Appendix A and give here a short summary of the 305 features in the set. They can be categorised into five groups:

- *features related to the problem size:*
  - concerning the scheduling period (3)
  - concerning the workforce (3)
- *feature related to the coverage requirements* (17)
- *features related to the workforce structure* (15)
- *features related to the contract constraints:*
  - concerning constraint values (45)
  - ratios (minimum over maximum constraint values) (12)
  - special ratios concerning the tightness of constraints (9)
  - concerning boolean constraints (4)
  - concerning the occurrence of patters in a contract (16)
- *features related to the requests* (56)

Note that all soft constraints can be given weights. The feature set thus includes both a weighted and an unweighted variant of features related to soft constraints.[11]

[11] This explains why the quantities in the list do not add up to 305.

In this larger study, we do not consider the SAT feature set. The results of the proof-of-concept study indicated that the SAT features were never more informative than the NRP features. In some cases, adding SAT features even decreased the accuracy of the prediction models. Moreover, the translation to SAT instances produced very large SAT instances for this instance distribution (over 300MB per instance), which made the use of SAT features rather impractical.

Step 4: Data generation

When building prediction models, a considerable amount of data must be available. The data used in the 2010 INRC contains only 15 instances in the medium track (on which the instance distribution is based). Consequently, we had to build a random instance generator producing similar instances in this distribution. We sample this generator and create a dataset of 800 random instances. This dataset is randomly partitioned into a training set of 600 instances and a validation set of 200 instances.

As we have mentioned earlier, we will be considering the average performance of the metaheuristics over five independent runs, in order to minimise the effect of random decisions on the solution quality. Our data generation step thus includes running the two considered algorithms, five times each, on all 800 instances in the dataset. We hereby record the average performance in terms of the quality of the obtained solutions. The stopping criterion of the algorithms is set at an equivalent of 10 minutes of calculation time. This data generation step obviously requires a large amount of computation time. Hence, we have parallellised the execution of this task and employed the cluster infrastructure of the VSC - Flemish Supercomputer Center.[12]

The second part of the data generation phase is the calculation and the processing of the feature values. Feature calculation is easily done by iterating over all concepts in the problem instances, and counting certain values. As the feature set is very extensive, and certain concepts are not really varied within the instance distribution, it is expected that certain values will be uni-valued or perfectly correlated to others. These features should be left out before any models are built. The feature set is designed for general problems in the formulation of Section 3.2. The set thus includes features looking at e.g. the number of skill types or the number of contract types. As these values are not varied across the dataset, these features will be uni-valued, and hence deleted. All features regarding soft constraints have both a weighted and an unweighted

[12]The VSC is funded by the Hercules foundation and the Flemish Government - department EWI. More information at https://www.vscentrum.be/

version. Since all weights are set to 1, these features will be perfectly correlated. We will thus discard the weighted versions. After removing uni-valued and perfectly correlated features, the remaining set contains 129 potentially useful features. These features are present in all five groups described above.

Note that the fact that certain values are not varied over the instances does not mean that this instance distribution is not of practical use. Indeed, in many real-world organisations, there will also be a limited number of contract types, skill types or shift types. Any real-world application using performance predictions will be focusing on the specifics of its own context. This will inevitably result in set of uni-valued or perfectly correlated features.

Step 5: Building performance prediction models

As for the proof-of-concept study, we employ various machine learning techniques made available through the WEKA software tool (Hall et al., 2009). We start by building performance prediction models using all of the 129 available features. We have built performance prediction models based on the training set and evaluate them using 10-fold cross-validation on the training set. The best models are further evaluated on the validation set of unseen instances in order to determine their true performance on unseen data.

Table 3.9 summarises the results of various models learned using all 129 features. It shows the correlation coefficients of a subset of the techniques that we have applied. There are two parts in the table, corresponding to the results for `Algorithm A` and `Algorithm B` respectively. The two columns show the correlation of the models when evaluated using 10-fold cross-validation on the training set and when evaluated on the validation set of unseen instances.

For both algorithms, the M5P Tree model is the most accurate, although some other models achieve similar correlation coefficients. These high correlation coefficients indicate good predictive power of the models. Indeed, when the regression trees are evaluated on the validation set of unseen instances, the correlation coefficients are still very high: $R = 0.992$ and $R = 0.994$ for `Algorithm A` and `Algorithm B` respectively. Figure 3.5 shows a plot of the actual versus the predicted quality of the solutions of `Algorithm A` on the validation set of unseen instances. Figure 3.6 shows a similar plot for `Algorithm B`. All data points lie close to the diagonal, confirming the high accuracy of the predictions.

However, as argued in Section 2.2.2, smaller models should be preferred. We will thus investigate whether smaller models can achieve similar results too, or even better results. We could use a number of different feature selection

		cross-validation	validation set
`Algorithm A`	Linear Regression	0.948	0.955
	Grid Search	0.942	0.956
	REP Tree	0.958	0.965
	M5P Tree	**0.988**	0.992
	Decision Table	0.963	0.959
	M5 Rules	0.985	0.989
`Algorithm B`	Linear Regression	0.950	0.957
	REP Tree	0.958	0.967
	M5P Tree	**0.990**	0.994
	Decision Table	0.964	0.959
	M5 Rules	0.981	0.991

Table 3.9: Correlation coefficients (R) of the various models based on all 129 features.

techniques. WEKA offers both learner-independent and learner-dependent techniques. One of the easiest and fastest techniques is correlation-based feature selection. This technique selects a set of features with high correlation to the target value, while trying to minimise the correlation between selected features. We experimented with forward, backward and bi-directional correlation-based feature selection and built prediction models based on the resulting feature sets. The forward correlation-based feature selection approach for `Algorithm A` selected 13 features, the backward approach 19 features and the bi-directional approach resulted in the same set as the forward approach. For `Algorithm B`, the forward approach selected 13 features, the backward approach 21 features and the bi-directional approach 14 features. These resulting feature sets are explicitly listed in Appendix B (in Tables B.1−B.5). Using these feature sets, we built a second set of prediction models using various techniques.

The results are summarised in Table 3.10. It shows the correlation coefficients of the resulting models based on the different feature selection approaches, evaluated using 10-fold cross-validation on the training set. It appears that the correlation-based feature selection approach is not leading to accurate models. The only exception here is the feature set resulting from the backward feature selection approach for `Algorithm B`. Using this feature set, several machine learning techniques were able to produce models with a high correlation

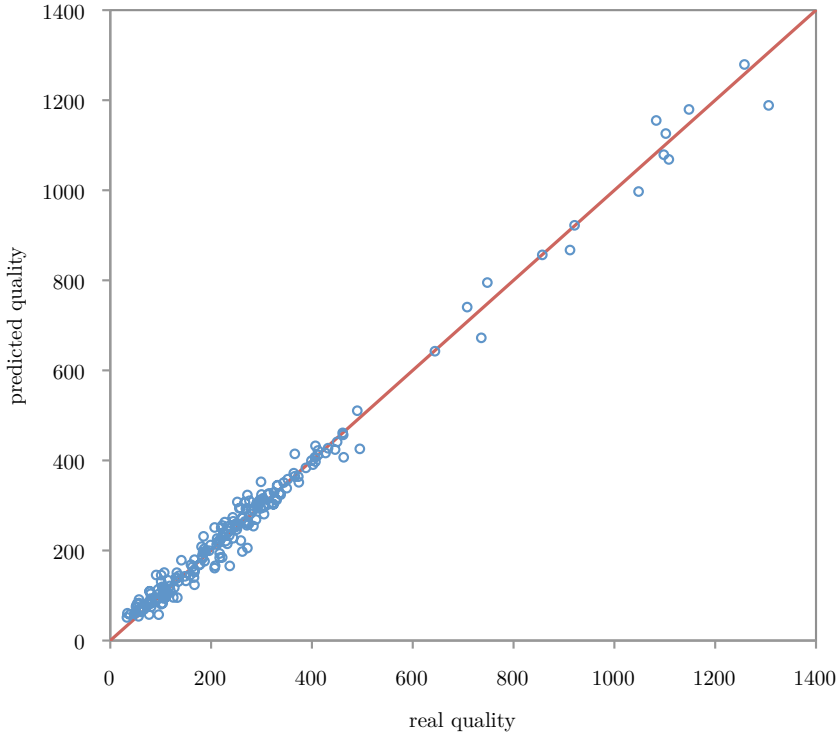


Figure 3.5: Actual versus predicted solution quality on the validation set for the M5P Tree model for `Algorithm A`.

coefficient (e.g. $R = 0.98$ for the M5P Tree model). It is not clear why only this specific feature set is leading to good results. We will further look into this in the following subsection. At this point, we will first investigate whether other feature selection techniques lead to better results.

Using WEKA, we applied as set of learner-dependent feature selection techniques. Such techniques iteratively build prediction models and select the features based on the quality of these models. The choice for which type of model to use is arbitrary, but in order to be feasible, the construction time for these models should not be too long. Linear regression models fit in well, at least for the forward selection approach. Backward learner-dependent feature selection requires building too many large prediction models and turns out to be taking too long (i.e. it does not finish in over two hours).

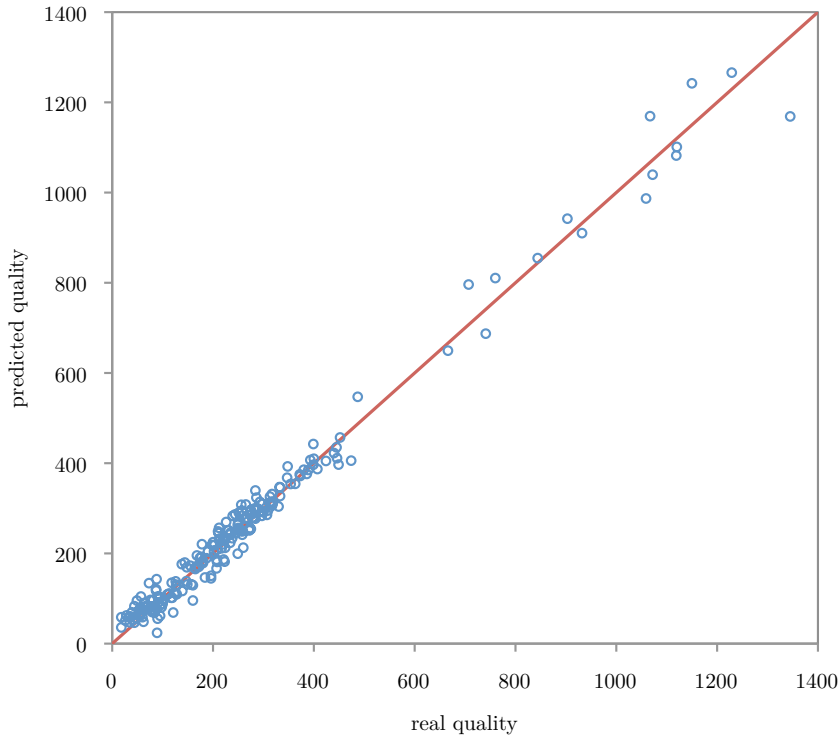


Figure 3.6: Actual versus predicted solution quality on the validation set for the M5P Tree model for `Algorithm B`.

When applied for `Algorithm A` and `Algorithm B`, the reduced sets contain 15 and 25 features respectively. These sets are given in Tables B.6 and B.7 respectively. Using these reduced feature sets, we built another set of performance prediction models. The results are summarised in Table 3.12. In contrast to the smaller models resulting from correlation-based feature selection, those based on linear regression generally lead to highly accurate performance predictions. As for the larger models, regression trees are among the best performing methods. Figure 3.7 shows a plot of the actual versus the predicted quality of the solutions of `Algorithm A` on the validation set of unseen instances. Predictions were made using the Multilayer Perceptron model (which is an artificial neural network approach) highlighted in Table 3.12. Figure 3.8 shows a similar plot for the M5P Tree model for `Algorithm B`.

Comparing the prediction models based on the reduced feature sets to those

		forward	backward	bi-directional
Algorithm A	Linear Regression	0.7370	0.7892	0.7370
	Multilayer Perceptron	0.6436	0.6300	0.6436
	Decision Table	0.6207	0.6773	0.6207
	M5 Rules	0.7509	0.7462	0.7509
	M5P Tree	0.7656	0.7892	0.7656
Algorithm B	Linear Regression	0.7334	0.7319	0.7321
	Multilayer Perceptron	0.5013	0.9548	0.6146
	Decision Table	0.6176	0.9630	0.6167
	M5 Rules	0.7221	0.9773	0.7483
	M5P Tree	0.7614	0.9838	0.7690

Table 3.10: Correlation coefficients (R) of the various models based on the reduced feature sets, evaluated using 10-fold cross-validation on the training set.

based on all 129 available features, we find no significant differences. The prediction accuracy is only marginally lowered and not significantly worse. The feature set, on the other hand, is drastically reduced. Tables 3.9 and 3.12 and the graphs in Figures 3.5−3.6 and 3.7−3.8 show that there is no clear distinction between the larger and smaller models in terms of prediction accuracy. As a consequence, the smaller models should thus be preferred over the larger models.

Investigating the reduced feature sets

As mentioned above, the correlation-based feature selection approach did not lead to accurate prediction models, the exception being the backward selected set presented in Table B.4. When we compare this set to the other correlation-based reduced feature sets (in Tables B.3 and B.5), we see that a number of features are present in all selected sets: three features related to the requests, three features related to the contract constraints, and two features related to the coverage constraints. In some sets, up to five additional features related to contract constraints, and two features related to the coverage constraints are selected. There are always three to eight features related to the structural contract constraints present, as well as two to four features related to the coverage constraints. The sets based on forward and bi-directional approaches

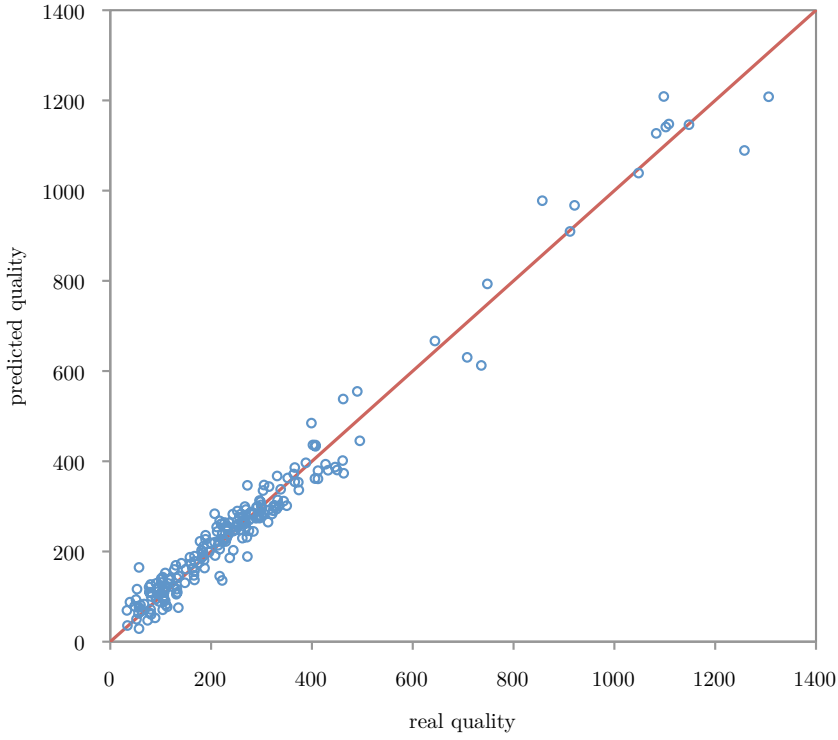


Figure 3.7: Actual versus predicted solution quality on the validation set for the Multilayer Perceptron model for `Algorithm A`.

also contain one feature related to the skills: the variation over the nurses of the number of skills they posses.

When comparing the feature sets based on the backward approach for both algorithms, we see that the set for `Algorithm B` is composed of all features in the respective set for `Algorithm A`, augmented with two additional features. The difference in accuracy of the resulting models must thus be caused by missing these two features. The first one is a contract constraint related feature (the minimum over all contract types of the minimum number of consecutive working weekends); the second one a coverage constraint related feature (the `nrNursesPerShift`: the number of nurses divided by the total number of shifts that need to be worked). Of these two features, only the last one seems to be really important. Indeed, when the first feature is removed from the set, the correlation coefficient of the M5P Tree model does not change. The model

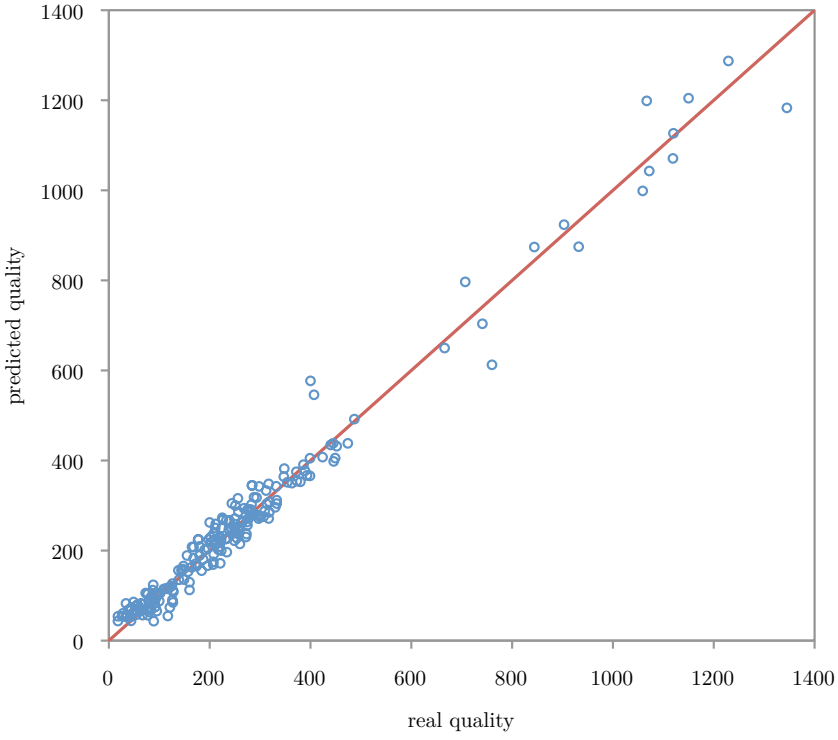


Figure 3.8: Actual versus predicted solution quality on the validation set for the M5P Tree model for `Algorithm B`.

remains highly accurate. However, when the coverage constraint related feature is removed from the set, the correlation coefficient drops significantly to $R = 0.76$. We also inverted this process, and added the `nrNursesPerShift` feature to the other feature sets. The correlation coefficients of various models based on these extended feature sets are shown in Table 3.11. We see a significant increase in prediction accuracy of the learned models, which proves the crucial importance of this feature.

Although we have now established the importance of this particular feature, this was not a straightforward result of the feature selection procedure. Simple correlation-based feature selection was in most cases not including this feature in the resulting set, making this technique not the best choice for reducing the feature set. It is not at all clear why this is the case.

		forward + 1	backward + 1	bi-directional + 1
`Algorithm A`	Linear Regression	0.7370	0.7390	0.7370
	Multilayer Perceptron	0.9593	0.9527	0.9593
	Decision Table	0.9651	0.9649	0.9651
	M5 Rules	0.9779	0.9741	0.9779
	M5P Tree	0.9857	0.9855	0.9857
`Algorithm B`	Linear Regression	0.7334	0.7319	0.7321
	Multilayer Perceptron	0.9593	0.9548	0.9517
	Decision Table	0.9631	0.9630	0.9631
	M5 Rules	0.9764	0.9773	0.9767
	M5P Tree	0.9841	0.9838	0.9844

Table 3.11: Correlation coefficients (R) of the various models based on the reduced feature sets augmented with the `nrNursesPerShift` feature, evaluated using 10-fold cross-validation on the training set.

		cross-validation	validation set
`Algorithm A`	Linear Regression	0.9340	0.9397
	Multilayer Perceptron	**0.9882**	0.9863
	Decision Table	0.9633	0.9582
	M5 Rules	0.9826	0.9690
	M5P Tree	0.9874	0.9857
`Algorithm B`	Linear Regression	0.9353	0.9374
	Multilayer Perceptron	0.9859	0.9918
	Decision Table	0.9615	0.9555
	M5 Rules	0.9875	0.9757
	M5P Tree	**0.9880**	0.9880

Table 3.12: Correlation coefficients (R) of the various models based on the reduced feature sets.

When we applied the learner-dependent feature selection technique based on linear regression models, the resulting feature sets did lead to highly accurate prediction models. To our surprise however, the `nrNursesPerShift` feature was not included in these sets. Instead, a similar and highly correlated feature is included: the `ratioAvailabilityOverCoverage`. This feature represents the maximum number of shifts that can be worked by all nurses divided by the total number of shifts that need to be covered. It thus even more precisely expresses a tightness measure for the constraints in the instances.

The correlation of both features can easily be explained. Since the nurses work according to one of four contract types, and the distribution of nurses over the contract types is fixed, the number of nurses is strongly correlated to the number of shifts that can be worked. Figure 3.9 shows a plot of the `nrNursesPerShift` feature versus the `ratioAvailabilityOverCoverage` feature. The correlation coefficient is $R = 0.99$.

While the importance of the `ratioAvailabilityOverCoverage` feature might seem obvious to domain experts, it is not at all clear how it really influences the quality of the solutions. One might expect that the higher this ratio is, the easier the problem will be to solve, and that the critical value will be somewhere around 1. In this region, all nurses need to work their maximum number of shifts to cover all the work that needs to be done. Any value lower than 1 would lead to an over-constrained problem where the constraint violations are much higher. This basic reasoning can also be observed in the considered instance distribution. Figure 3.10 shows a plot of the `ratioAvailabilityOverCoverage` feature values versus the solution quality of `Algorithm A`. It is clearly visible that there is a steep downward trend up to a ratio of 1, after which there is a much milder upward trend. The left part of the figure corresponds to the over-constrained region. As expected, the cost quickly increases as ratio drops further below 1. On the other hand, the upward trend on the right part is somewhat surprising. The number of constraint violations increases as more and more nurses become available. Having more than enough nurses appears to be problematic, and should thus also be avoided. This can be explained by their minimum work requirements. These constraints specify that all nurses should work at least a certain number of shifts. As more nurse become available, not all of them will be working enough to satisfy these requirements. Note that this ratio is related to the *total coverage constrainedness* as defined by Maenhout (2007). In his PhD dissertation, Maenhout introduces this indicator as a parameter for the generation of nurse rostering instances.

Figure 3.10 also shows a critical value of the ratio of 0.728. There is a clear separation of the data around this value. All instances with a ratio higher than 0.728 have a weighted sum of constraint violations lower than 660. All instances with a ratio lower than 0.728 have a higher weighted sum of constraint

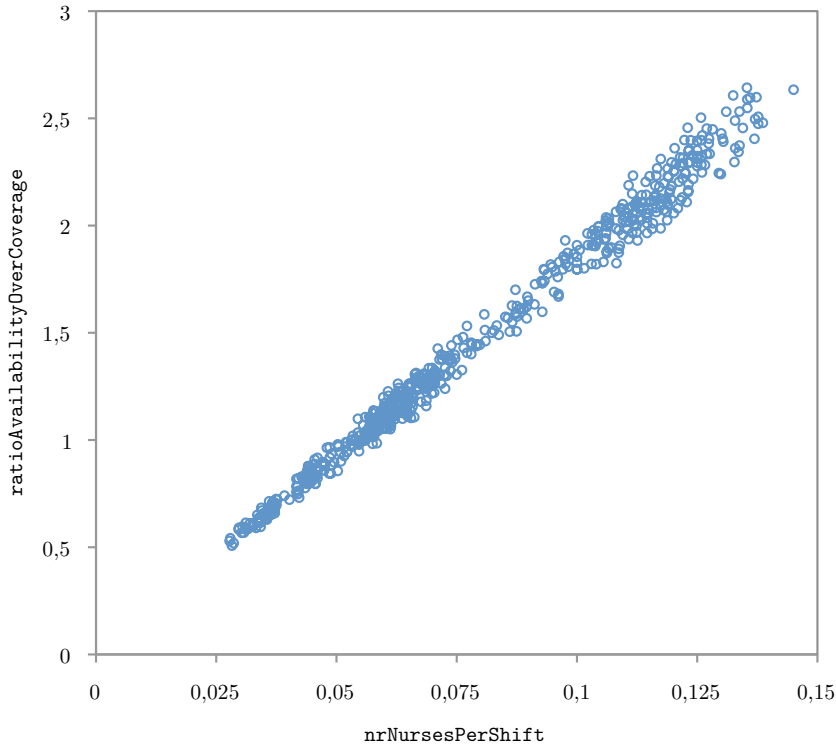


Figure 3.9: The `nrNursesPerShift` feature versus the `ratioAvailability OverCoverage` feature.

violations. There are no data points in the other quadrants of the graph in Figure 3.10. This indicates that when the ratio drops below 1, i.e. when nurses are required to work more than they are allowed to, the overall solution quality is not considerably affected. At least not at first; only when this ratio drops even more, the weighted number of constraint violations starts to explode. The weighted number of constraint violations is thus higher when this ratio is below this threshold.

As argued in Section 2.2.2, smaller prediction models have a higher potential for experts to interpret or investigate the relationship between features and performance. Indeed, when we examine the models in more detail, we get an indication of the relative importance of certain features. Figure 3.11 shows an outtake of the M5P Tree model for the prediction of the performance of

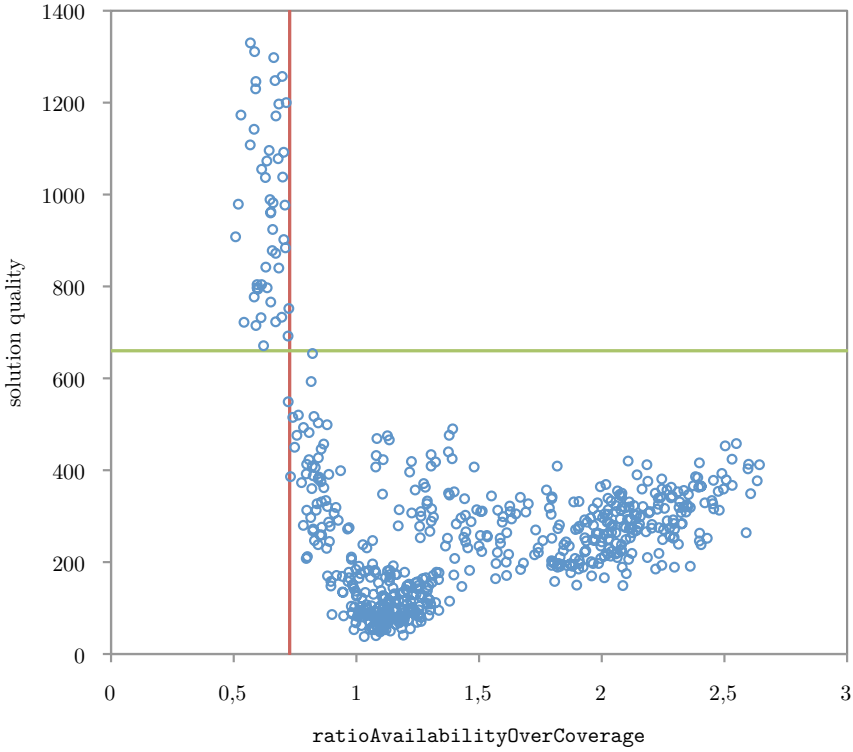


Figure 3.10: `ratioAvailabilityOverCoverage` versus the solution quality of `Algorithm A`.

`Algorithm A`, based on the reduced feature set. The decision tree and one of the regression models at the leaf level are shown. Immediately, the importance of the `ratioAvailabilityOverCoverage` feature is confirmed. The top two levels of the tree are split based on the value of this feature. The main decisions in this regression tree are thus based on this important feature. Moreover, when looking at the actual linear regression models at the leaves in the tree, these models have high coefficients associated to this feature. Figure 3.11 only shows one regression model, but the coefficients for this feature in the other leaves range from 4.43 to 1342.48. Note that all feature values have been normalised before the model construction. Higher regression coefficients (in case of normalised feature values) could indicate a higher relative importance with respect to other features. However, caution must be taken with such statements, as two perfectly correlated features could appear with equal and arbitrary large

coefficients but with opposite signs in a regression model, without having any effect on the performance of the model.

Having shown that feature selection techniques can greatly reduce the feature sets without affecting the prediction accuracy much, we grew interested in how much further we could take this process. A correlation analysis shows that certain features in the reduced sets for `Algorithm A` are strongly correlated to others. We thus deleted the following features:

- the maximum number of required nurses per skill type (strongly correlated to the number of shifts that need to be covered)
- the maximum number of nurses per contract type (strongly correlated to the standard deviation of the number of nurses per contract type)
- the maximum number of required nurses per shift type (strongly correlated to the standard deviation of the number of required nurses per shift type)

The regression tree model based on this further reduced feature set achieves the same correlation to the solution quality as the model in Figure 3.11. The decision tree is identical, only the regression models are different. Note that for these correlated features, the choice of which feature to keep and which one to delete is arbitrary. We tried both options and the resulting correlation coefficients are not different. The coefficients associated to the `ratioAvailabilityOverCoverage` feature are still among the higher ones.

We furthermore re-applied learner-dependent feature selection techniques to the feature set in order to further reduce the set. Given that this set contained only 13 features, we could employ more time-consuming machine learning techniques in the learner-dependent feature selection approach. We applied the decision tree learners (M5P Tree) in a forward selection step, resulting in a set of 9 features. The M5P Tree model based on these 9 features achieves a correlation of $R = 0.9874$ based on 10-fold cross-validation on the training set. When we further remove features, one by one, starting with those having the least negative effect on prediction correlation, we end up with a feature set containing only three features. Using only these three features, an M5P Tree model for `Algorithm A` achieves a correlation coefficient of $R = 0.9797$, using 10-fold cross-validation on the training set. For `Algorithm B`, the regression tree model achieves a correlation coefficient of $R = 0.9855$ when evaluated using 10-fold cross-validation on the training set. Further removing any of these three features leads to a larger decrease in prediction accuracy (i.e. the correlations drop to values around $R = 0.74$). The remaining three features for both algorithms are the following:

```
M5P Tree:
ratioAvailabilityOverCoverage <= 0.221 :
|   ratioAvailabilityOverCoverage <= 0.103 : LM1
|   ratioAvailabilityOverCoverage >  0.103 : LM2
ratioAvailabilityOverCoverage >  0.221 :
|   ratioAvailabilityOverCoverage <= 0.373 :
|   |   requiredNursesPerShift_stddev <= 0.309 :
|   |   |   MinConsecutiveFreeDays_mean <= 0.394 :
|   |   |   |   requiredNursesPerShift_stddev <= 0.238 :
|   |   |   |   |   MinConsecutiveFreeDays_mean <= 0.285 : LM3
|   |   |   |   |   MinConsecutiveFreeDays_mean >  0.285 :
|   |   |   |   |   |   ratioAvailabilityOverCoverage <= 0.242 : LM4
|   |   |   |   |   |   ratioAvailabilityOverCoverage >  0.242 : LM5
|   |   |   |   requiredNursesPerShift_stddev >  0.238 : LM6
|   |   |   MinConsecutiveFreeDays_mean >  0.394 :
|   |   |   |   ratioAvailabilityOverCoverage <= 0.275 : LM7
|   |   |   |   ratioAvailabilityOverCoverage >  0.275 :
|   |   |   |   |   MaxNumAssignments_mean <= 0.422 : LM8
|   |   |   |   |   MaxNumAssignments_mean >  0.422 :
|   |   |   |   |   |   nrShifts <= 0.261 : LM9
|   |   |   |   |   |   nrShifts >  0.261 :
|   |   |   |   |   |   |   requestsShiftOffPerNurse_variation <= 0.448 : LM10
|   |   |   |   |   |   |   requestsShiftOffPerNurse_variation >  0.448 : LM11
|   |   requiredNursesPerShift_stddev >  0.309 :
|   |   |   requiredNursesPerShift_stddev <= 0.441 : LM12
|   |   |   requiredNursesPerShift_stddev >  0.441 : LM13
|   ratioAvailabilityOverCoverage >  0.373 :
|   |   nursesPerContract_stddev <= 0.446 : LM14
|   |   nursesPerContract_stddev >  0.446 : LM15

LM num: 1
quality =
         1116.0369 * nrShifts
        - 189.5459 * requiredNursesPerShift_stddev
        +  21.7103 * requiredNursesPerSkill_stddev
        + 207.7034 * requiredNursesPerSkill_maximum
        +  33.2998 * nursesPerContract_stddev
        -  68.7281 * MaxNumAssignments_mean
        - 144.8266 * MaxConsecutiveWorkingDays_mean
        +  82.1364 * MinConsecutiveFreeDays_mean
        - 327.7504 * ratioAvailabilityOverCoverage
        +   7.1684 * requestsOffPerNurse_minimum
        -  10.3341 * requestsShiftOffPerNurse_variation
        -  12.4098 * requestOffPerDay_maximum
        +  30.4667 * requestsOffPerShift_variation
        + 265.0183

<other regression models omitted>
```

Figure 3.11: Outtake of the M5P Tree model for the prediction of the performance of `Algorithm A`, based on the reduced feature set.

- for `Algorithm A`:
  - `ratioAvailabilityOverCoverage`: the maximal number of shifts that can be worked by the nurses divided by the total number of shifts that need to be covered
  - `MinConsecutiveFreeDays_mean`: the mean value over all contract types of the minimum number of consecutive free days
  - `requiredNursesPerShift_stddev`: the standard deviation over all shifts of the required number of nurses
- for `Algorithm B`:
  - `ratioAvailabilityOverCoverage`: the maximal number of shifts that can be worked by the nurses divided by the total number of shifts that need to be covered
  - `MinConsecutiveFreeDays_mean`: the mean value over all contract types of the minimum number of consecutive free days
  - `requiredNursesPerDay_maximum`: the maximum number of nurses that are required on any given day in the planning horizon

When evaluated on the validation set of unseen instances, the correlation coefficients are $R = 0.9784$ and $R = 0.9838$, for the models predicting the quality of the solutions obtained by `Algorithm A` and `Algorithm B` respectively. Figure 3.12 presents an outtake of the regression tree model for `Algorithm A`, based on only three features. Figures 3.13 and 3.14 show plots of the predictions of the regression tree models versus the actual solution quality of `Algorithm A` and `Algorithm B` respectively, evaluated on the validation set of unseen instances. These figures show that the predictions are indeed fairly accurate. The correlation is high, but the prediction errors have increased with respect to the models based on larger feature sets.

It is rather remarkable that the quality of the solutions can be predicted based on only three instance features. For we had already discovered the importance of the `ratioAvailabilityOverCoverage` feature, this further empirical investigation proves that this feature is even more important than already thought. Together with the `requiredNursesPerDay_maximum` feature, these can be considered as measures of the load on the personnel. The `ratioAvailabilityOver Coverage` feature counts the available resources per unit of requested coverage and the `requiredNursesPerDay_maximum` gives an indication of what size of staff is required to be. The `MinConsecutiveFreeDays_mean` feature, on the other hand, is a structural feature influencing the flexibility in the staff while the `requiredNursesPerShift_stddev` feature has something to say about the variation in the demand. The fact that such features are needed for good predictions relates to particular structural properties of nurse rostering problems.

```
M5P Tree:
ratioAvailabilityOverCoverage <= 0.98 :
|   ratioAvailabilityOverCoverage <= 0.728 : LM1
|   ratioAvailabilityOverCoverage >  0.728 : LM2
ratioAvailabilityOverCoverage >  0.98 :
|   ratioAvailabilityOverCoverage <= 1.303 :
|   |   requiredNursesPerShift_stddev <= 3.244 :
|   |   |   MinConsecutiveFreeDays_mean <= 4.338 :
|   |   |   |   requiredNursesPerShift_stddev <= 2.94 :
|   |   |   |   |   MinConsecutiveFreeDays_mean <= 3.549 : LM3
|   |   |   |   |   MinConsecutiveFreeDays_mean >  3.549 :
|   |   |   |   |   |   ratioAvailabilityOverCoverage <= 1.024 : LM4
|   |   |   |   |   |   ratioAvailabilityOverCoverage >  1.024 : LM5
|   |   |   |   requiredNursesPerShift_stddev >  2.94 : LM6
|   |   |   MinConsecutiveFreeDays_mean >  4.338 : LM7
|   |   requiredNursesPerShift_stddev >  3.244 :
|   |   |   requiredNursesPerShift_stddev <= 3.805 : LM8
|   |   |   requiredNursesPerShift_stddev >  3.805 : LM9
|   ratioAvailabilityOverCoverage >  1.303 : LM10

LM num: 1
quality =
          265.687  * requiredNursesPerShift_stddev
        +   28.8376 * MinConsecutiveFreeDays_mean
        -   62.9315 * ratioAvailabilityOverCoverage
        -  371.818

LM num: 2
quality =
          127.0752 * requiredNursesPerShift_stddev
        +   53.1069 * MinConsecutiveFreeDays_mean
        - 1118.9485 * ratioAvailabilityOverCoverage
        +  632.2946

LM num: 3
quality =
           41.6902 * requiredNursesPerShift_stddev
        +  10.9910 * MinConsecutiveFreeDays_mean
        -   0.0870 * ratioAvailabilityOverCoverage
        -  67.1223

LM num: 4
quality =
           31.7317 * requiredNursesPerShift_stddev
        +  11.7891 * MinConsecutiveFreeDays_mean
        -  28.1267 * ratioAvailabilityOverCoverage
        -   1.6492

<other regression models omitted>
```

Figure 3.12: Outtake of the M5P Tree model for the prediction of the performance of `Algorithm A`, based on only three features.

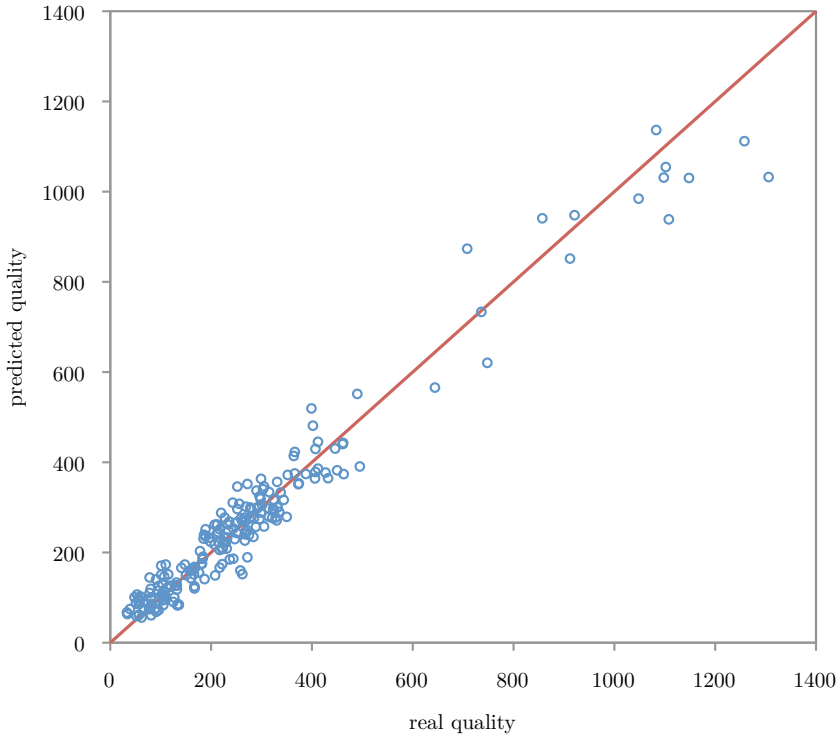


Figure 3.13: Actual versus predicted solution quality on the validation set for the M5P Tree model for `Algorithm A` based on three features.

3.4 Applications

We have shown in the previous section that we can build accurate performance prediction models for two state-of-the-art algorithms for the nurse rostering problem (in the formulation of the 2010 INRC). Predicting the output of such algorithms opens several perspectives for their application. The type of algorithms for which we have built predictors is fit for real-world applications. In this section, we discuss a number of applications of such prediction models in practical settings. First, we focus on algorithm selection in Section 3.4.1. Afterwards, we discuss a number of other applications in Section 3.4.2.

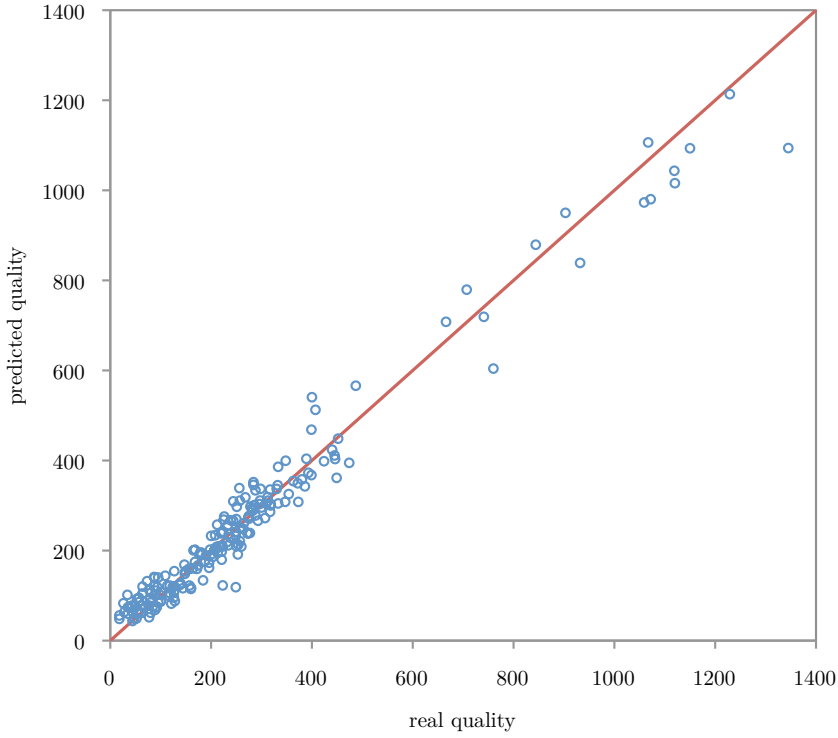


Figure 3.14: Actual versus predicted solution quality on the validation set for the M5P Tree model for `Algorithm B` based on three features.

3.4.1 An algorithm selection tool for nurse rostering

When looking at benchmarks, often no single algorithm outperforms all others. Each algorithm has its own strengths and weaknesses. When computing resources are scarce, and instances need to be solved quickly, selecting the best algorithm for a given instance is the way to go.

In the context of the larger experiment, we are dealing with two state-of-the-art algorithms solving nurse rostering problems with practical relevance. This setting could potentially be interesting for building a portfolio solver, given that the algorithms are indeed *competitive*. Through the proposed framework in Section 2.3.1, we can check whether this is indeed the case.

When considering the complete dataset, we find a competitiveness ratio of only

	always-A	always-B	AS1	AS2	AS*
% correctly classified instances	5.5	94.5	86.5	97.0	100
avg. rel. dev. from best (%)	10.67	0.08	1.00	0.05	0
avg. quality (cost)	276.14	264.29	265.13	264.06	263.56

Table 3.13: Performance of different algorithm selection strategies on the validation set.

$c = 0.1500$. This is rather low, indicating that an algorithm selection tool will have to be very accurate. The low competitiveness is a result from a very low equipotency $e = 0.1511$. Indeed, when we compare the performance of both algorithms, it seems to be the case that `Algorithm A` is outperforming `Algorithm B` on only 7.5% of the instances. There are only few instances on which both algorithms perform equally, leading to a high reach $r = 0.9925$, but due to the low equipotency, the resulting competitiveness is also very low. The difference in performance is furthermore small. The resulting potential impact is only $i = 0.28\%$. In other words, `Algorithm B` is the best choice for 92.5% of the instances and for those instances where it is outperformed by `Algorithm A`, the difference is relatively small. Consequently, an algorithm selection approach will have to be extremely accurate to be able to improve over the best single-algorithm strategy (i.e. always applying `Algorithm B`). The measures thus indicate that there is little hope for big improvements when combining both algorithm in a portfolio solver. Nevertheless, sometimes a small improvement is already much appreciated.

In a first attempt at constructing such a portfolio, we simply apply the performance prediction models of the previous section. When a new instance needs to be solved, the solution quality for both algorithms is predicted and the best algorithm is run. The results of this approach (denoted as `AS1`) are summarised in Table 3.13. This table also includes the results of always selecting `Algorithm A` (denoted as `always-A`) and always selecting `Algorithm B` (denoted as `always-B`)). The optimal case (where the best algorithm is always chosen, denoted as `AS*`) is also included. These values are calculated based on the same validation set as used in the previous section (thus only containing instances which were not used for training). As can be seen from this table, the results of the portfolio solver are not better than always selecting `Algorithm B`. A straightforward application of the performance prediction models developed in this chapter does not allow for a sufficiently accurate selection approach.

There are however other options when building algorithm selection tools. Instead of making two separate quality predictions and comparing these values, one can also try to predict the algorithm choice directly. We have therefore added an extra attribute to the training data, namely the algorithm which performed best. In case of a tie, we left this value empty. Using various classification techniques in WEKA, we constructed a set of prediction models for this attribute. When presented with a new instance, the selection approach (denoted as `AS2`) now calculates the necessary features and directly predicts an algorithm choice. This algorithm is then run on the instance. The results of this approach are also included in Table 3.13. It contains the results of the best performing classifier, which is based on the RandomForest model of WEKA. As can be seen from Table 3.13, `AS2` is outperforming `always-B` on all criteria.

The competitiveness ratio indicated that a successful selection approach would require highly accurate decisions. `AS2` is able to select the right algorithm for 97% of instances, which is indeed very high. Furthermore, the average relative deviation from the best solution is only 0.05%, which indicates that `AS2` is indeed a very successful algorithm selection approach.

3.4.2 Other applications

In this section, we describe a number of possible application domains for performance prediction models, other than algorithm selection tools.

Automated negotiation schemes require a cost estimate of a specific scenario. This estimate needs to be reliable but not necessarily very accurate. The negotiation scheme should not take long to evaluate. In the case of nurse rostering, one can think of a system ruling the exchange of nurses between wards in a hospital or from an exchange pool. When considering calling in a nurse from the pool, internal rearrangements normally produce several possibilities. These possibilities are negotiated among the wards taking into account the local cost/benefit expressed by the objective function. It is not always feasible to exactly evaluate all these possibilities, as this involves solving many nurse rostering instances. Fast estimates of the solution quality an algorithm can achieve are needed. Performance prediction models can provide such estimates (See e.g. Haspeslagh et al., 2007, 2013; Haspeslagh and De Causmaecker, 2013).

Another application can be found in patient admission scheduling (Vancroonenburg et al., 2013; Bilgin et al., 2012). An optimal patient admission policy is essential for the financial sanity of hospitals and it impacts the performance indicators used in hospital evaluation. Assigning patients to specific rooms in a hospital influences the coverage requirements of the wards and the workload of the nurses. It can thus be useful to take into account these effects when

scheduling patient admissions. A reliable predictor for algorithm performance can support fast decision making in patient admission and could increase the hospital's overall capacity utilisation.

3.5 Conclusions

In this chapter, we have undertaken an experimental study investigating the feasibility of building accurate performance prediction models for nurse rostering problems.

The literature provides us with many problem formulations and related solution methods for nurse rostering problems. In this dissertation, we focused on the formulation of the 2010 International Nurse Rostering Competition. Our experimental study consists of two phases. First, we carried out a proof-of-concept study in which we determined the feasibility of building accurate performance prediction models for nurse rostering. In the second phase, we extended the results of the initial experiments to larger and more complex problem instances with practical relevance.

In the initial proof-of-concept study, the instance distribution is limited, allowing the application of both complete search methods and metaheuristic methods. We investigated the use of two distinct feature sets. The first feature set consists of domain-specific features, based on the properties varied within the instance distribution. The set is relatively small, containing only 11 features focusing specifically on the considered instance distribution. Its potential value for other distributions is rather limited. The second feature set is a subset of an existing and more general feature set developed for propositional satisfiability problems. In order to compute these features for nurse rostering instances, we employed an efficient translation scheme transforming nurse rostering instances into SAT instances. We generally observed that using the SAT feature set is not improving the results of using only the NRP feature set. Neither as an alternative, nor as an addition to the NRP features. In many cases, the SAT feature set was able to produce similar results; but the main disadvantage of using such features is the requirement of a translation of the instances to SAT problems.

The proof-of-concept instance distribution allowed to consider both complete and incomplete algorithms. We looked at a complete integer programming solver and a simple local search method. In this context, we investigated the prediction of three distinct performance criteria: running time of the complete solver, quality of the solutions of both algorithms, and the quality gap between the approximate and optimal solution.

In contrast to earlier work on decision problems, we did not succeed in accurately predicting the running time of the complete solver, nor the logarithm of this running time, regardless of the feature set. Alternatively, predicting whether the running time is above or below some threshold was successful. With respect to the quality of the solutions however, we were also able to produce highly accurate prediction models, using either of the two feature sets. This is one of the main results of the proof-of-concept study, demonstrating the feasibility of performance predictions for a practical combinatorial optimisation problem being solved by a metaheuristic.

In the second part of this study, we focused on realistic nurse rostering instances with practical relevance. This required designing an extended feature set, characterising instances in terms of 305 feature values. This feature set is one of the main contributions of this chapter. We have employed this feature set for the prediction of the empirical hardness of two state-of-the-art algorithms, ranked top-five in the 2010 INRC. The resulting models are highly accurate. As smaller models are generally preferred, we applied several feature selection techniques. We found that simple correlation-based feature selection did not always lead to good results. The application of a learner-dependent selection approach based on linear regressions models did however produce qualitative results. For both algorithms, the reduced feature set led to highly accurate prediction models.

While investigating the performance prediction models, we discovered that one particular feature is of crucial importance for the success of prediction models. We found that the regressions trees were highly dependent on the `ratioAvailabilityOverCoverage` feature. This feature represents the number of shifts that can be covered by all nurses in the pool, divided by the number of shifts that need to be covered during the scheduling period. Eliminating this feature from the set significantly reduces the prediction accuracy.

After deeper investigation, we found the most important features to be related to the workload imposed on the nurses (i.e. the `ratioAvailabilityOverCoverage` feature) and to the variation in the structural constraints (i.e. the `MinConsecutiveFreeDays_mean` feature) and the demand (i.e. the `requiredNursesPerShift_stddev` and the `requiredNursesPerDay_maximum` features). The analysis of prediction models provides relevant insight into which features are most important.

We have successfully built an application of performance prediction in the form of an automatic algorithm selection tool. The considered algorithms were only poorly competitive on the instance set, leading to poor results when straightforwardly applying the resulting prediction models. However, when classification tools are employed building models directly predicting

the best performing algorithm, highly accurate results were found. The resulting algorithm selection tool was able to outperform any of its components individually.

In this chapter, we have thus demonstrated the feasibility and practical use of performance predictions in the context of nurse rostering problems.

Chapter 4

Multi-mode resource-constrained project scheduling

The field of project management is sometimes described as a discipline of planning, organising, timing, allocating and controlling resources (Demeulemeester and Herroelen, 2002). The main goal is to achieve a number of performance, cost and time objectives while efficiently using a set of resources. Projects are divided into a number of tasks. Tasks are linked together by temporal constraints indicating that some tasks must be completed before others can start. The execution of tasks requires resources, which can be scarce. The primary challenge of project management is to achieve all of the project's goals and objectives, while satisfying the constraints. Three main activities fall within the scope of project management: (1) planning, (2) scheduling and (3) controlling. The planning phase concerns the specification of the tasks to be performed. The resource requirements, duration and cost estimates are bundled together. During the project scheduling phase, an execution plan is constructed for the project's tasks. The challenge is to optimise this plan according to some predetermined objective like the minimisation of the execution time or the economical resource usage. This includes deciding on an execution order for the tasks and optimising the allocation of the resources. The control phase takes places once the actual execution of tasks has commenced and deals with errors or unforeseen circumstances.

In this chapter, we will focus on the project scheduling problem. We consider the

requirements to be given and look at algorithms building execution schedules. We start by giving a definition of the problem, including a mathematical model, in Section 4.1. In Section 4.2, we review some relevant literature on project scheduling and discuss efforts in quantifying the complexity of the problem. We extensively discuss our experimental study towards algorithm performance prediction for project scheduling problems in Section 4.3. In Section 4.4, we discuss applications of performance prediction models. One of the main contributions will be an automatic algorithm selection tool outperforming any of its components. In Section 4.5, we draw conclusions from and formulate some interesting directions for further research.

4.1 Problem definition

This section describes the multi-mode resource-constrained project scheduling problem in detail. We start with a definition of the basic resource-constrained project scheduling problem (RCPSP), which is then extended to include multiple execution modes and a new resource type to form the multi-mode resource-constrained project scheduling problem (MRCPSP).

The resource-constrained project scheduling problem is a classic problem formulation for project scheduling. In its basic form, the resource-constrained project scheduling problem can be described as follows. The RCPSP considers a project consisting of a number of activities. Each activity has a predefined processing time (or duration). The order in which the activities of the project should be executed is determined by a set of precedence constraints. Each activity has a number of predecessors, i.e. a set of activities that must be completed before this activity can start. The precedence constraints can be represented by a directed graph (also known as an activity network). Each activity is represented by a node and there are directed arcs from each activity to its immediate successors. It is assumed that this graph is acyclic. A number of renewable resource types constrain the problem. Each resource type has a certain capacity indicating the maximum number of units of this resource type that can be used concurrently at any given time. Each activity requires a certain amount of units of a number of resource types for its execution. The fact that resources are considered to be renewable implies that their capacity is constant over time. After an activity has finished, the required resource units are released and can be reused by another activity. A schedule is an assignment of start times to the activities. For a schedule to be feasible, the precedence constraints, as well as the resource constraints should be satisfied. The objective is to find a feasible schedule with minimal makespan, i.e. a schedule for which the last activity finishes as early as possible. It is common to add two dummy

activities to the problem formulation, representing the start and the finish of a project. Additional precedence constraints state that the start activity is the predecessor of all activities that do not have any other predecessors, and that the end activity has as immediate predecessors all activities that are not an immediate predecessor to any other activity. These dummy activities have a duration of zero time units and require no resource units. A schedule with minimal makespan then corresponds to a schedule with minimal start time for the dummy end activity. Blazewicz et al. (1983) have shown that the RCPSP is a strongly NP-hard optimisation problem.

While this definition is already a rich model, several extensions have been formulated for coping with different situations occurring in practice. One commonly used variant is the multi-mode resource-constraint project scheduling problem. The MRCPSP is like a regular RCPSP, but the activities can be processed in several different modes. Each mode corresponds to a combination of a processing time and a set of resource requirements. Modes thus represent different manners of executing an activity. When considering multiple modes, a new resource type is introduced: the non-renewable resource. Non-renewable resources represent concepts that disappear after they have been consumed. Hence, they are not available any more for any of the following activities. Money and raw materials are examples of non-renewable resources. Machine power, on the other hand, is an example of a renewable resource type. Non-renewable resource types also have a predetermined capacity. An activity mode thus specifies a duration and a set of resource requirements for both the renewable and the non-renewable resource types. A schedule for the MRCPSP is an assignment of start times and mode selections to the activities. The objective is again to find a schedule with minimal makespan. There exist other objectives too (see e.g. the survey of Hartmann and Briskorn, 2010), but minimal makespan is one of the most commonly used criteria. Note that there may not be a feasible solution in the presence of non-renewable resources. Moreover, as shown by Kolisch and Drexl (1997), this feasibility problem is NP-complete when at least two non-renewable resources are present. The MRCPSP can be classified as $m, 1T|cpm, disc, mu|C_{max}$ in the classification scheme of Herroelen et al. (1998) or as $MPS|prec|C_{max}$ following the classification scheme of Brucker et al. (1999).

4.1.1 Mathematical model

There are several models available for formally expressing the MRCPSP. We present an exact description of the problem as a mixed integer program, based on the model of Kolisch and Sprecher (1996), which is similar to the model of Talbot (1982). The choice for this model is based on its widespread use

J	:	The number of activities, labelled $j = 1, \ldots, J$, including a dummy start and end activity.
M_j	:	The number of execution modes of activity j.
R	:	The set of renewable resource types.
N	:	The set of non-renewable resource types.
$\bar{T}$	:	An upper bound on the project's makespan, calculated by sequentially executing all activities in their longest mode.
C_r^R	:	The number of available units (i.e. the capacity) of the renewable resource type $r \in R$.
C_r^N	:	The capacity of the non-renewable resource type $r \in N$.
$k_{j,m,r}^R$	:	The number of units of renewable resource type $r \in R$ that is needed to execute activity j in mode m.
$k_{j,m,r}^N$	:	The number of units of non-renewable resource type $r \in N$ that is needed to execute activity j in mode m.
$d_{j,m}$	:	The processing time or duration of mode m of activity j.
P_j (S_j)	:	The set of immediate predecessors (successors) of activity j.
ES_j (EF_j)	:	The earliest start (finish) time of activity j, calculated using minimal durations and neglecting resource usage.
LS_j (LF_j)	:	The latest start (finish) time of activity j, calculated using minimal durations and neglecting resource usage, but counting backwards from the upper bound $\bar{T}$.

Table 4.1: Symbols and definitions for the mathematical model of the MRCPSP.

throughout the research communities. Part of its success follows from the public availability of a large benchmark set (Kolisch and Sprecher, 1996).

We introduce the notation in Table 4.1. The objective is to minimise the total makespan of the project. This is equivalent to minimising the start time of the dummy end activity (labelled J). The model hence minimises the following sum (4.1) subject to the linear expressions in (4.2 - 4.6).

$$\text{MINIMISE} \sum_{t=ES_J}^{LS_J} t.x_{J,1,t} \tag{4.1}$$

The binary decision variables $x_{j,m,t}$ are defined in Expression (4.2). They denote whether activity j is performed in mode m and started at time t.

$$\text{for } j = 1, \ldots, J; \quad m = 1, \ldots, M_j; \quad t = ES_j, \ldots, LS_j :$$

$$x_{j,m,t} = \begin{cases} 1 \text{ if activity } j \text{ is performed in mode } m \text{ and started at time } t \\ 0 \text{ otherwise} \end{cases} \tag{4.2}$$

The constraint in Expression (4.3) ensures that each activity is assigned to exactly one mode and exactly one starting time.

$$\text{for } j = 1, \ldots, J : \quad \sum_{m=1}^{M_j} \sum_{t=ES_j}^{LS_j} x_{j,m,t} = 1 \tag{4.3}$$

Expression (4.4) resembles the precedence constraints. It assures that all of j's predecessors $i \in P_j$ are finished before the execution of j starts. ($d_{i,m}$ denotes the processing time of job i in mode m.)

$$\text{for } j = 1, \ldots, J; \quad \forall i \in P_j :$$

$$\sum_{m=1}^{M_i} \sum_{t=ES_i}^{LS_i} (t + d_{i,m}) x_{i,m,t} \leq \sum_{m=1}^{M_j} \sum_{t=ES_j}^{LS_j} t.x_{j,m,t} \tag{4.4}$$

The constraints in Expressions (4.5) and (4.6) ensure the resource requirements for the renewable and non-renewable resource types respectively.

$$\forall r \in R; \quad \text{for } t = 1, \ldots, \bar{T} :$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} k_{j,m,r}^{R} \sum_{q=\max(t-d_{j,m}, ES_j)}^{\min(t-1, LS_j)} x_{j,m,q} \leq C_r^R \tag{4.5}$$

$$\forall r \in N :$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} k_{j,m,r}^{N} \sum_{t=ES_j}^{LS_j} x_{j,m,t} \leq C_r^N \tag{4.6}$$

4.2 Literature overview

The project scheduling problem originates from the fields of economics and management. Over the years, it has become an important subject in operational research too. In the scientific literature, the resource-constrained project scheduling problem (RCPSP) has become the standard problem formulation for project scheduling. There are many real-world application areas for project scheduling. Only a few examples are in large production facilities or concern the building and construction sector. In general, most companies, both large and small, have to deal with some form of project management tasks. As we have stated earlier, the RCPSP is a powerful and widely used paradigm, yet several adaptations or extensions have been proposed throughout the years, dealing with additional constraints present in specific real-world cases. In this chapter, we focus on the multi-mode resource-constrained project scheduling problem, where tasks are to be executed in one of several modes, with different durations and different resource requirements. Project scheduling algorithms must thus both select the best execution modes and decide on an execution scheme (i.e. the start times of the activities) satisfying the precedence and resource constraints.

Throughout the years, several complete methods have been proposed for the MRCPSP. Examples are presented by Talbot (1982), Sprecher et al. (1997) and Zhu et al. (2006). Because these algorithms aim at finding an optimal solution, they are not very suitable for large problem instances since calculation times would be too long. For that matter, heuristic and metaheuristic approaches have gained importance in the community. Such methods produce good solutions in a limited amount of time. There is however no guarantee that the final result is an optimal solution. We refer the reader to Hartmann and Briskorn (2010) for an extensive survey on solution methods (both complete and incomplete methods) for variants and extensions of the resource-constrained project scheduling problem. Węglarz et al. (2011) survey single-project, single-objective, deterministic project scheduling problems where activities can be scheduled in a finite or infinite number of modes. More recently, Van Peteghem and Vanhoucke (2014) presented an overview of current state-of-the-art metaheuristic algorithms for the multi-mode resource-constrained project scheduling problem.

They evaluated these algorithms on a newly presented benchmark set. This benchmark set will later also be used in this dissertation.

4.2.1 On the complexity of project scheduling

In this section, we review a number of important contributions concerning the complexity of project scheduling. We summarise a number of characteristics that have been linked to the problem difficulty of project scheduling. Such complexity indicators are important in the context of performance predictions. Although these characteristics were originally investigated for single-mode problems, we will use them as an inspiration for building an extensive feature set for multi-mode problems.

During the last decades of the previous century, investigating such characteristics for problems represented by an activity network was a very lively topic. Several measures were proposed and the relationships between these indicator values and the complexity of the instances were empirically determined. In the resulting literature, researchers mainly considered the intrinsic complexity of problem instances, independent of the algorithm being used. This contrasts to the empirical hardness in which we are interested in this dissertation. Empirical hardness however includes the intrinsic hardness of the instances. The complexity indicators will thus probably be important for building performance prediction models too. Few studies investigated the relationship of instance properties and the performance of certain algorithms. Davis (1975) and Patterson (1976) studied the effects of problem structure on the performance of complete search algorithms for problems using an activity network representation. Elmaghraby and Herroelen (1980) found a relationship between complexity measures and the algorithm being used. A network considered complex for one algorithm might be easy for another algorithm. While, at that time, researchers were mostly interested in the algorithm independent complexity of the problems, Elmaghraby and Herroelen (1980) already stated that it is unlikely that the difficulty, or complexity, of a network can be captured by only one measure.

A number of complexity measures have been proposed in the literature. In this section, we will summarise the definitions of these indicators. They will serve as a basis for building an instance feature set for the (multi-mode) resource-constrained project scheduling problem. The indicators mainly concern the size of the network, the topological structure and the availability of resources.

- The *coefficient of network complexity* (*CNC*), introduced by Pascoe (1966) is defined as the ratio of the number of arcs over the number of nodes in an activity network. It was first used for activity-on-the-arc networks

and later also for activity-on-the-node networks. In the latter context, Kolisch et al. (1995) observed that a more complex problem instance has more connections in the network, leading to a higher value of the *CNC*. However, subsequent studies seem to confirm that instances become easier as the *CNC* increases, because there is less freedom in deciding on an order to schedule the activities in, which is contradictory to the result of Kolisch et al. (1995). The *CNC* clearly contains some information on the complexity of project scheduling instances. It is however not sufficient to discriminate between networks that have an equal number of nodes and arcs, but very different degrees of complexity.

- The *order strength* (*OS*), defined as the number of precedence relations divided by the theoretical maximum number of such relations ($n(n-1)/2$, with n the number of activities), was first introduced by Mastor (1970). It is also referred to as the *density* by Kao and Queyranne (1982) and, as observed by Elmaghraby and Herroelen (1980), the order strength is equal to 1 minus the *flexibility ratio* of Dar-El (1973). The *OS* thus represents the degree of constrainedness with respect to the precedence constraints.

- The *reduction complexity* as defined by Bein et al. (1992) is the minimum number of node reductions needed to reduce a two-terminal acyclic network to a single edge. It has later been redefined as the the *complexity index* (*CI*) by De Reyck and Herroelen (1996). These authors also conclude that the *CI* is more informative than the *CNC*, but it is still not sufficient to accurately discriminate between hard and easy RCPSP instances. This is somewhat intuitive, as these measures do not take into account any information on the requirements and availabilities of the resources. In terms of practical usefulness, this indicator has the disadvantage that its computation is not straightforward. We refer to De Reyck and Herroelen (1996) for a detailed description and an algorithm to compute the *CI*.

- The *resource factor* (*RF*), introduced by Pascoe (1966), reflects the degree to which activities request units of the different resource types. It is defined as the average over the activities of the number of resource types for which units are requested by the activity over the total number of resource types present in the problem definition. If all activities request units of all resource types, then the *RF* equals 1.

- The *resource strength* (RS_k) of a (renewable) resource type k was first introduced by Cooper (1976) as the capacity of a resource type k divided by the average number of units requested by the activities. It is similar to the resource factor, but regards resource units instead of resource types. Because of this, the resource strength is defined for each resource type. Kolisch et al. (1995) have redefined this measure because of some

limitations of the earlier definition. In its original formulation, the resource strength was not normalised to the interval $[0, 1]$. It was also easy to generate two problem instances with equal RS, but with very different complexity (Kolisch et al., 1995). In its later definition (Kolisch et al., 1995; Kolisch, 1996), the resource strength is a normalised value expressing the relationship between the resource requirements and the resource availability and taking into account some information on the precedence constraints. As for the complexity index, calculation of this version of the resource strength is not straightforward. We refer the reader to Kolisch et al. (1995) for a detailed description of the resource strength.

- The *resource constrainedness* (RC_k) of a (renewable) resource type k was introduced by Patterson (1976) and is defined as the average demand of k (i.e. the average number of units requested by all activities) divided by the availability (i.e. the capacity of the resource type). It is thus related to the earlier definition of the resource strength (Cooper, 1976). It can be seen as a normalised version of this resource strength. There exist arguments for using the resource constrainedness instead of the resource strength. First, the RC is a more 'pure' availability measure as it does not incorporate information on the precedence constraints. Second, there are occasions where the RS can no longer discriminate between easy and hard instances while the RC continues to do so (Patterson, 1976).

In their study on phase transitions in project scheduling, Herroelen and De Reyck (1999) conclude that there is not yet a totally unambiguous measure for the resource availability. In this chapter, we will propose a feature set for the characterisation of project scheduling problem instances. This set covers most of the indicators discussed above. We describe this set in detail in Appendix C.

The above mentioned measures are static, they are independent of a solution method and can be calculated a priori. As an alternative to these measures, it is also possible to look at characteristics of (partial) solutions. Such properties are observed while an algorithm is solving an instance. For example, when a number of activities have already been scheduled, it is possible to look at resource scarceness measures. Such measures relate to the complexity of scheduling the remaining activities. They can be used to guide an algorithm in completing a partial schedule. Buddhakulsomsiri and Kim (2007) propose a *moving resource strength*, which helps their priority rule-based heuristic for the MRCPSP where activity splitting is allowed. Van Peteghem and Vanhoucke (2011) use a *resource scarceness matrix* to decide which improvement step will be taken by their scatter search algorithm in the next iteration. This matrix looks at the scarceness of both renewable and non-renewable resources, given the mode assignments of an individual. The idea is that when the resource scarceness is

low, the final schedule will be mostly determined by the precedence constraints and hence, the algorithm should focus on this. When resources are scarce on the other hand, the resource constraints will become more important and the algorithm should focus on these constraints. Overall, such characteristics can only be calculated for (partial) solutions, and can help a procedure *while* it is solving an instance. It is however not possible to calculate such measures a priori, which is necessary for any prediction model that is to be applied without running the actual algorithm.

4.3 Performance prediction for project scheduling

In this section, we focus on the experimental study towards performance prediction for the multi-mode resource-constrained project scheduling problem. We will present this study along the same lines as we did for the nurse rostering case in the previous chapter. We will thus follow the same five-step procedure for building algorithm performance prediction models.

Step 1: Instance distribution

The most commonly used benchmark set for project scheduling is the publicly available PSPLib library,[1] introduced by Kolisch and Sprecher (1996). Initially, this library contained benchmark sets for both the RCPSP and MRCPSP. Since its publication, the library has been extended with several benchmark sets containing instances for other extensions proposed in the literature. Throughout the years, researchers have been using these benchmarks for evaluating their algorithms. By the time of writing this dissertation, over 15 years have passed since the publication the first benchmark instances. Due to significant advances in both algorithmic design and computer infrastructure, we can currently consider the benchmark instances as being rather small and out-dated. Current state-of-the-art algorithms generally produce optimal or near-optimal solutions for most benchmark instances. Even in the worst case, the obtained solutions are never far from optimal (i.e. the optimality gap is below one or two percent). In terms of performance on this benchmark set, there is little room for improvement for new state-of-the-art algorithms.

When the PSPLib instances are analysed in terms of order strength and resource strength, it appears that there is little variation in the set. In his PhD dissertation, and later in a journal article, Van Peteghem makes similar observations and proposes a new benchmark library called MMLib

[1] The PSPLib benchmark instances can be found at http://129.187.106.231/psplib/.

(Van Peteghem, 2010; Van Peteghem and Vanhoucke, 2014).[2] The set contains multi-mode instances that are considerably larger and more diverse, in terms of order strength, resource factor and resource strength. The newly proposed benchmark set consists of three subsets:

- MMLIB50 contains 540 instances with projects consisting of 50 activities. There are always three possible execution modes. The order strength, resource strength and resource factor are varied.
- MMLIB100 contains 540 instances with projects consisting of 100 activities. As for the previous set, there are always three execution modes and the order strength, resource strength and resource factor are varied.
- MMLIB+ contains 3240 instances with projects consisting of 50 or 100 activities. Activities have either three, six or nine execution modes with varying order strength and resource strength (the resource factor is always 1 for these instances).

Current state-of-the-art algorithms still perform well on average on these instances, as is shown by Van Peteghem and Vanhoucke (2014). However, when looking at the per-instance performance, there are larger differences between algorithms, as will be shown next. These differences make this benchmark set particularly interesting for a study towards empirical hardness and automatic algorithm selection. In this chapter, we will therefore focus on the MMLIB benchmark set for building performance prediction models.

Step 2: Algorithm set

We will investigate algorithm performance prediction models for two different state-of-the-art algorithms. The choice for these algorithms is based on the availability of executable code. More algorithms could be included, but this is considered to be future work.

The first algorithm considered in this study, denoted as `Algorithm A`, is the tabu search algorithm of Nonobe and Ibaraki (2002). This metaheuristic was not specially designed for the MRCPSP. It can handle more general resource-constrained project scheduling problems for other extensions as well. Solutions are represented by the commonly used activity list representation (See e.g. Alvarez-Valdes and Tamarit, 1989; Nonobe and Ibaraki, 2002; Lova et al., 2009). Each solution is hereby represented by an activity list and a mode list. The

[2]The MMLIB benchmark instances can be found at `http://www.projectmanagement.ugent.be/mrcpsp.html`.

activity list is an array specifying the order in which the activities are to be scheduled. This order satisfies the precedence constraints. The mode list is also an array, and specifies the mode in which each activity is to be scheduled. The tabu search algorithm uses a method very similar to the standard serial scheduling scheme to build a schedule from an activity list (Alvarez-Valdes and Tamarit, 1989). This method iterates over the activity list and schedules each activity as soon as possible, given the resource requirements. It thus defines a deterministic way of getting from the activity list representation to an actual schedule. The search space of the algorithm is defined by a set of moves generating neighbours of the current solution (i.e. the current activity and mode lists). The tabu search algorithm works its way through the search space, looking for better solutions. In order to escape from local optima, worsening moves are allowed as longs as this leads to a different area in the search space. The algorithm therefore remembers the recent moves (in a tabu list) and avoids getting back to previously visited regions. The tabu tenure (i.e. the length of the tabu list) is adaptively controlled based on certain properties of the search process. We refer the reader to Nonobe and Ibaraki (2002) for more details on this tabu search algorithm. This algorithm has been shown to be state-of-the-art on both the PSPLib instances (Nonobe and Ibaraki, 2002) and the MMLib benchmark set (Van Peteghem and Vanhoucke, 2014).

The second algorithm, further on denoted as `Algorithm B`, is a hybrid genetic algorithm implemented by ourselves, but based on the algorithm described by Lova et al. (2009). The core idea is to evolve a population of individuals, similar to the evolution of animals. Strong individuals have a higher probability of being combined (through a crossover combination of their genome) into new individuals, hoping to have inherited the good properties of the parent population. Diversification is included through random mutations. This genetic algorithm operates on gene strings, which are based on the same activity list representation as used by `Algorithm A`. Additionally, two extra bits are appended to the gene string: one bit indicating whether a serial or parallel scheduling scheme must be used for transforming the activity list into a schedule, the other bit indicating whether this scheduling scheme will be applied in a forward or backward manner. Doing so, the algorithm automatically learns which scheduling scheme to apply to which individuals. This idea was first introduced by Kolisch and Drexl (1996) and was also used by Hartmann (2002). The algorithm is further enhanced by adding an efficient mode improving step, turning it into a hybrid approach. This method takes the generated schedule of each individual and tries to shorten the makespan by changing the modes of certain activities. This step, as shown by Lova et al. (2009), is very effective in reducing the makespan on the resulting schedules. `Algorithm B` has been shown to be state-of-the-art on both the PSPLib benchmark set (Lova et al., 2009) and the MMLib benchmark set (Van Peteghem and Vanhoucke, 2014).

Table 4.2 summarises the performance of both algorithms on the PSPLIB benchmark set. It shows both the average relative deviation from the optimal (or best known) solution in percentages and the percentage of optimal solutions found. To allow for a fair comparison, both algorithms were stopped after generating 5000 schedules. Within the field of project scheduling, this is the commonly used way of testing and comparing heuristic algorithms on benchmark instances (See e.g. Hartmann and Kolisch, 1998; Van Peteghem and Vanhoucke, 2014). It has been observed that the generation of schedules based on activity lists is generally the most time-consuming step in the solution process. In contrast to using a strict time limit, the number of generated schedules is independent of programming skills or computer hardware, and thus a better stopping criterion. We see that `Algorithm B` outperforms `Algorithm A` on all subsets separately and on the combined set as well. However, bearing in mind that the maximal makespan in this benchmark set is only 66 time units, an average deviation from the optimal solution of less than 2% can still be considered a very good performance.[3] Both algorithms obtain optimal (or best known) solutions in at least 70% of the instances, and even when this is not the case, the solution is never far from optimal (or best known) in terms of quality (makespan). Moreover, the difference between both algorithms is never large. There is little room for improvement, and both algorithms can be considered state-of-the-art on this set.

When the algorithms are run on the larger MMLIB dataset, we get a different picture. Table 4.3 summarises the performance of both algorithms on the MMLIB benchmark set. The average performance over five independent runs is shown for two different stopping criteria: 5000 and 25000 schedules. Given the more complex nature of this benchmark set, it is natural to consider longer calculation times. The first line for each algorithm displays the average relative deviation from the best solution found by any of these two algorithms. The second line shows the average absolute deviation from the best solution. The last line shows the percentage of instances on which the corresponding algorithm performed better than or equal to the other algorithm. We see that the algorithms perform differently on the different subsets:

- When given a limit of 5000 schedules, `Algorithm A` turns out to be the overall better choice on the MMLIB50 and the MMLIB100 subsets. It is better in terms of the number of best solutions found and it achieves a lower average relative and absolute deviation from the best found solution than `Algorithm B`. However, on the MMLIB+ subset (and also on the combination of all benchmark sets), `Algorithm B` achieves the lowest

[3]The smallest possible deviation is 1 time unit; with a maximal makespan of 66 time units, as soon as there is a difference, this difference is at least 1.52%.

		instance set of PSPLIB							
		j10	j12	j14	j16	j18	j20	j30	all
`Algorithm A`	avg. rel. dev. from best (%)	1.19	1.17	1.31	1.48	1.78	2.55	3.40	1.84
	% best solutions found	87.0	84.6	78.8	74.9	64.1	62.2	48.0	71.3
`Algorithm B`	avg. rel. dev. from best (%)	0.07	0.14	0.45	0.76	1.01	1.50	2.97	0.99
	% best solutions found	99.5	96.7	90.6	84.3	74.1	70.8	57.8	81.9

Table 4.2: Comparison of the performance of both algorithms on PSPLIB benchmark instances.

		instance set of MMLIB			
		MMLIB50	MMLIB100	MMLIB+	all
`Algorithm A` (5000 schedules)	avg. rel. dev. from best (%)	0.87	1.19	2.38	2.09
	avg. abs. dev. from best	0.28	0.64	3.69	2.99
	% best solutions found	65.6	70.9	58.0	60.2
`Algorithm B` (5000 schedules)	avg. rel. dev. from best (%)	2.75	3.92	3.16	3.20
	avg. abs. dev. from best	1.08	1.68	2.70	2.41
	% best solutions found	55.1	46.2	44.8	46.1
`Algorithm A` (25000 schedules)	avg. rel. dev. from best (%)	0.63	0.21	0.32	0.34
	avg. abs. dev. from best	0.17	0.08	0.30	0.26
	% best solutions found	75.3	89.8	90.8	89.0
`Algorithm B` (25000 schedules)	avg. rel. dev. from best (%)	3.39	2.77	9.46	8.41
	avg. abs. dev. from best	1.37	5.87	10.20	8.43
	% best solutions found	48.9	33.3	11.1	17.6

Table 4.3: Comparison of the performance of both algorithms on MMLIB benchmark instances.

absolute deviation from the best solution. In case the ultimate goal is to find schedules with minimal makespan for a large number of instances, then the evaluation of algorithms should be based on the absolute deviation from the best solution, as it reflects this goal the best. `Algorithm A` might thus be the better choice for more instances; in terms of the achieved makespan, it is `Algorithm B` that generally leads to better solutions.

- When both algorithms are allowed to construct 25000 schedules, the results are different. `Algorithm A` has become more dominant over `Algorithm B`, being the best choice on 89% of the instances, regardless of the evaluation criterion being considered. It seems that `Algorithm A` benefits more from longer calculation times than `Algorithm B`. Nevertheless, `Algorithm B` is still outperforming `Algorithm A` on 11% of the instances.

As we will see later (in Section 4.4), the 5000 schedules case leads to a high competitiveness among these algorithms, making it an ideal sandbox for building automatic algorithm selection tools. This has furthermore supported the choice for considering these two algorithms in this study.

When dealing with metaheuristic algorithms, we need to look at other performance criteria than running time. In this study, the considered algorithms are stopped once a predetermined number of schedules is generated. This threshold is related to the running time, but independent of programming language, skills or computer infrastructure. As argued above, this type of stopping criterion allows for a fair comparison between the algorithms since they can use more or less the same computational effort. Similar to our study on nurse rostering problems in Chapter 3, we will look at the quality of the obtained solutions, given a fixed stopping criterion. The quality of the solutions will here be measured in terms of the makespan, i.e. the finishing time of the last activity. Other properties could also be considered (such as e.g. objectives based on earliness (or tardiness), or the minimisation of resource usage or a related cost function, or any trade-off between such objectives); but since the algorithms in this study aim at minimising the makespan, it only seems fair to measure their performance using this criterion. The effect of random decisions is minimised by taking the average performance over five independent runs. Such averages are representative for the real performance of an algorithm, as long as the variation in quality over these runs is relatively low. For both stopping criteria, the coefficient of variation[4] is on average 4%, which indicates that the variation is indeed relatively low. As a consequence, the average performance is a good measure for the actual performance of these algorithms on the considered benchmark set.

[4]The coefficient of variation is also known as the relative standard deviation and is defined as the standard deviation divided by the mean.

Step 3: Feature selection

For performance prediction models to be accurate, we need to be able to capture those properties of the instances that influence the empirical hardness. These properties should be easily representable by a vector of feature values. We have therefore constructed an extensive feature set for general multi-mode resource-constrained project scheduling problem instances. The set contains 686 instance features and is described in detail in Appendix C. The features can be roughly categorised into four groups:

- *features related to the problem size* (35)
- *features related to the resource constraints* (486)
- *features related to the precedence constraints* (21)
- *features related to the activity durations* (144)

The set covers a wide range of statistics on the availability of the resources. Most of the indicators discussed in Section 4.2.1 are (in some form) included in the set. We choose to exclude the resource strength (as defined by Kolisch et al. (1995)) from the feature set, partly because of the computational effort (i.e. it requires building a critical path schedule), but mainly because we believe that representing the availability of resources as a vector of many values is a much more detailed approach than using only one aggregate value. We furthermore hope that this will allow for a better discrimination between instances. Additionally, the complexity index is also excluded from the feature set, due to its rather time-consuming computation.

All features in the set are efficiently computable. It is just a matter of iterating over the problem description and counting certain values.

Step 4: Data generation

A sufficiently large set of training data must be available for building accurate algorithm performance prediction models. As decided in Step 1, we consider the MMLIB benchmark set. This set consists of 4320 instances, divided over three subsets and is sufficiently large.

We run both algorithms on all benchmark instances. For each of the considered stopping criteria (5000 and 25000 schedules), we record the average performance over five independent runs. We have employed the cluster infrastructure of the VSC (Flemish Supercomputer Center) for executing this task. For some of these

instances, one or both algorithms were unable to find a feasible solution. In that case, the average over the remaining successful runs is taken. Instances for which one of the algorithms could not find a feasible solution in any of the five runs are filtered out. As a result, a set of 4140 benchmark instances remains. This set is randomly partitioned into a training set of 3140 instances, and a validation set of 1000 instances.

The second part of the data generation step concerns the calculation of the feature values. This is an efficient process and requires at most a few hundred milliseconds per instance. Due to the structure of the instances, some features are uni-valued or perfectly correlated to others. Examples of uni-valued features are those related to doubly constrained resources. Such resources can be represented by the combination of a renewable and a non-renewable resource and are not present in the considered instance distribution. In total, 103 uni-valued features are removed. Additionally, a correlation analysis reports 242 redundant features, which are perfectly correlated to at least one other feature. Such correlated features are also filtered out. (The choice of which features are deleted and which feature stays is arbitrary.) Examples of such features are certain minimum-over-maximum ratios where either the minimum or the maximum is uni-valued. Another example is the mean number of predecessors per activity, which is in fact the same as the mean number of successors per activity. The remaining feature set contains 341 features.

Step 5: Building performance prediction models

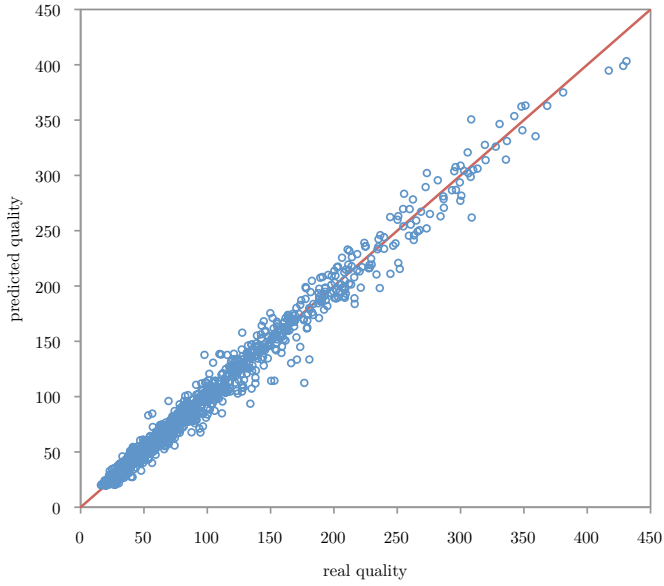
In this final step, given a sufficient amount of data being available, we construct a set of performance prediction models for the two algorithms selected in Step 2 above. As for the nurse rostering case of the previous chapter, we will first construct prediction models based on all available features. Subsequently, we will investigate whether a reduced feature set can achieve similar results. All prediction models are built using the training instances and evaluated based on 10-fold cross-validation on this training set. The best models are furthermore evaluated on the validation set in order to determine whether they are indeed performing well on unseen instances. We experiment again with a number of machine learning techniques made available through the Weka software tool of Hall et al. (2009).

The results of various models based on all available features are summarised in Table 4.4. It shows the correlation coefficients of the models when evaluated using 10-fold cross-validation on the training set. Furthermore, the table also shows the correlation coefficients for the models when evaluated on the validation set of unseen instances. For both algorithms, the M5-based models (both the

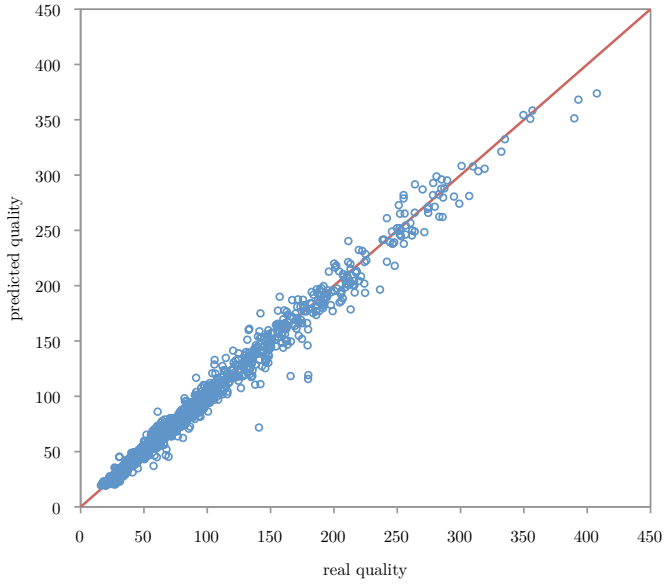
		cross-validation	validation set
`Algorithm A` (5000 schedules)	Linear Regression	0.9426	0.9444
	M5P Tree	0.9889	0.9914
	Decision Table	0.9237	0.9151
	M5 Rules	0.9880	0.9907
`Algorithm B` (5000 schedules)	Linear Regression	0.9474	0.9479
	M5P Tree	0.9892	0.9917
	Decision Table	0.9293	0.9150
	M5 Rules	0.9881	0.9907
`Algorithm A` (25000 schedules)	Linear Regression	0.9422	0.9430
	M5P Tree	0.9885	0.9908
	Decision Table	0.9244	0.9348
	M5 Rules	0.9882	0.9872
`Algorithm B` (25000 schedules)	Linear Regression	0.9416	0.9448
	M5P Tree	0.9885	0.9900
	Decision Table	0.9269	0.9146
	M5 Rules	0.9866	0.9885

Table 4.4: Correlation coefficients (R) of the various models predicting the solution quality, based on 341 features.

regression trees (M5P Tree) and rule sets (M5 Rules)) appear to be among the most accurate ones. For both stopping criteria, these models achieve correlation coefficients around $R = 0.99$ when evaluated using 10-fold cross-validation on the training set. These high correlation coefficients indicate good predictive power. Indeed, when the models are evaluated on the validation set of unseen instances, the correlation coefficients remain as high as $R = 0.99$. Figure 4.1 further confirms this statement. It shows the actual quality of the solutions versus the predicted quality by the M5P Tree models. The plots show the predictions on the validation set, (a) for `Algorithm A` and (b) for `Algorithm B` (both given a stopping criterion of 5000 schedules).



(a) `Algorithm A` (5000 schedules)

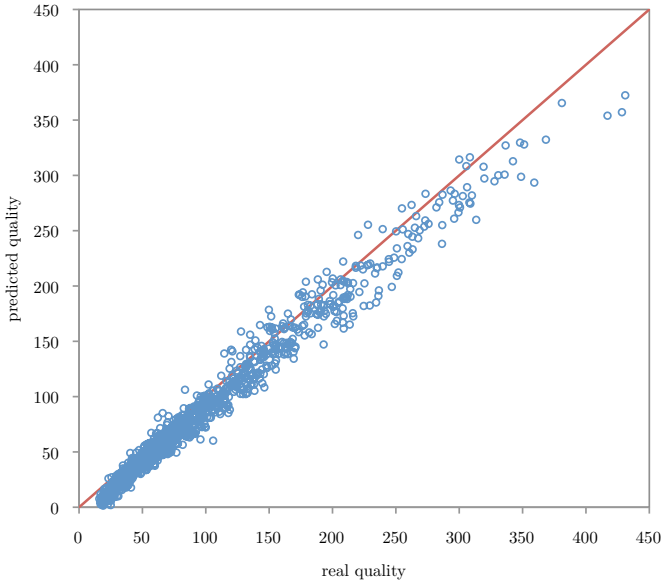


(b) `Algorithm B` (5000 schedules)

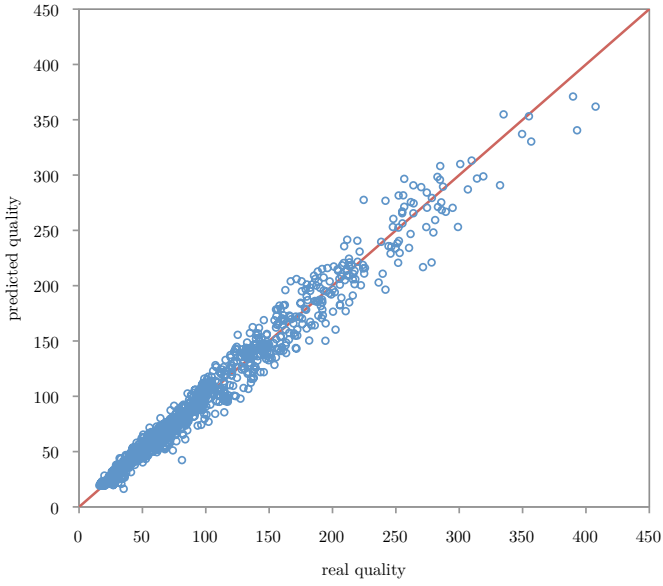
Figure 4.1: Actual versus predicted solution quality on the validation set for the M5P Tree model.

As we have argued in Section 2.2.2, prediction models based on fewer features should be preferred over larger models as long as their performance is not significantly worse. The experiments of the previous chapter furthermore showed that for nurse rostering algorithms, feature selection techniques could effectively be applied, leading to smaller models with similar accuracy as when the complete feature set is used. Consequently, we will apply a number of feature selection techniques reducing the size of the feature set and build prediction models based on these smaller sets for both project scheduling algorithms. We start with applying simple correlation-based feature selection techniques. We experiment with forward, backward and bi-directional approaches for both algorithms. The resulting feature sets contain between 24 and 45 features and are explicitly enumerated in Tables D.1 − D.8 in Appendix D. These sets contain mainly features related to the resource constraints. A smaller number of features are related to the precedence constraints and the activity durations. No strict size-related features have been selected by the correlation-based procedures. Large parts of these feature sets overlap, indicating that certain features appear to be important for both algorithms, regardless of the stopping criterion being imposed. The common features are: (1) features related to the minimum over maximum ratio of the requested resource units per execution mode, (2) a number of ratios of requested resources over the availabilities, (3) the number of precedence constraints over the theoretical maximum (which is defined as the *order strength* in Section 4.2.1), (4) the variation of the number of predecessors per activity and (5) features related to the minimum and variation of the duration of the activity modes.

Table 4.5 summarises the results of the prediction models based on the reduced feature sets. As can be seen, the M5-based models and the artificial neural networks are among the best performing methods, achieving correlation coefficients similar to those of the best performing models based on all available features ($R = 0.98$). In almost all cases, the backward correlation-based feature selection process leads to slightly better prediction models, and should thus be preferred over the forward selection process. The backward selected feature sets are generally larger (up to twice the size) than those selected by a forward selection process. The additional features are mostly related to the resource constraints. Figure 4.2 shows plots of the actual quality versus the predicted quality by the Multilayer Perceptron models built using the reduced feature sets based on backward correlation-based feature selection. The plots show the predictions on the validation set, (a) for `Algorithm A` and (b) for `Algorithm B` (both given the stopping criterion of 5000 schedules). Comparing this to Figure 4.1, we can see that the predictions are still highly accurate. The feature selection process did not have a negative impact on the quality of the prediction models. We find similar results in the setting where the algorithms are allowed to construct 25000 schedules.



(a) `Algorithm A` (5000 schedules)



(b) `Algorithm B` (25000 schedules)

Figure 4.2: Actual versus predicted solution quality on the validation set for the Multilayer Perceptron models based on the reduced feature sets (backward correlation-based feature selection).

		cross-validation forward	cross-validation backward	validation set forward	validation set backward
`Algorithm A` (5000 schedules)	Linear Regression	0.9129	0.9235	0.9151	0.9279
	Multil. Perceptron	0.9822	0.9888	0.9889	0.9931
	Decision Table	0.8647	0.8932	0.8916	0.9136
	M5 Rules	0.9773	0.9875	0.9831	0.9885
	M5P Tree	0.9800	0.9885	0.9839	0.9928
`Algorithm B` (5000 schedules)	Linear Regression	0.9232	0.9303	0.9237	0.9321
	Multil. Perceptron	0.9797	0.9905	0.9859	0.9939
	Decision Table	0.9002	0.8820	0.9259	0.9086
	M5 Rules	0.9802	0.9869	0.9836	0.9911
	M5P Tree	0.9827	0.9905	0.9870	0.9929
`Algorithm A` (25000 schedules)	Linear Regression	0.9137	0.9231	0.9136	0.9258
	Multil. Perceptron	0.9791	0.9876	0.9861	0.9915
	Decision Table	0.8560	0.8933	0.8801	0.9133
	M5 Rules	0.9760	0.9862	0.9752	0.9892
	M5P Tree	0.9764	0.9870	0.9756	0.9888
`Algorithm B` (25000 schedules)	Linear Regression	0.9148	0.9232	0.9175	0.9254
	Multil. Perceptron	0.9792	0.9907	0.9840	0.9937
	Decision Table	0.8788	0.8819	0.8896	0.9095
	M5 Rules	0.9797	0.9862	0.9797	0.9909
	M5P Tree	0.9815	0.9893	0.9826	0.9923

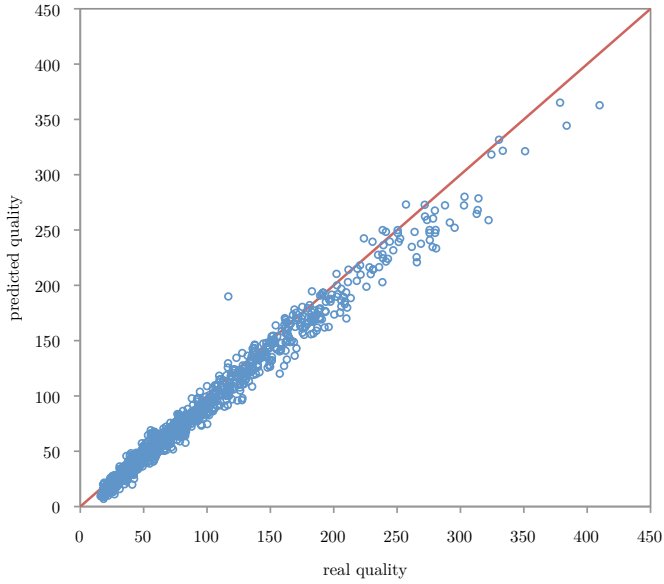
Table 4.5: Correlation coefficients (R) of the various models based on the reduced feature sets.

We furthermore experimented with more elaborate feature selection techniques and applied a forward, learner-dependent feature selection process using linear regression models. Given the size of the training set, such an approach would require too much time if being applied to the complete training set. In each iteration, prediction models would need to be trained using all instances. Instead, we have randomly selected 20% of the training instances and have applied the feature selection approach on this smaller training set. For the stopping criterion of 5000 schedules, this led to 25 and 27 features for `Algorithm A` and `Algorithm B` respectively. For the stopping criterion of 25000 schedules, this resulted in sets of 28 and 14 features for `Algorithm A` and `Algorithm B` respectively. These feature sets are enumerated explicitly in Tables D.9 – D.12 in Appendix D. In contrast to the correlation-based feature selection approaches, these feature sets

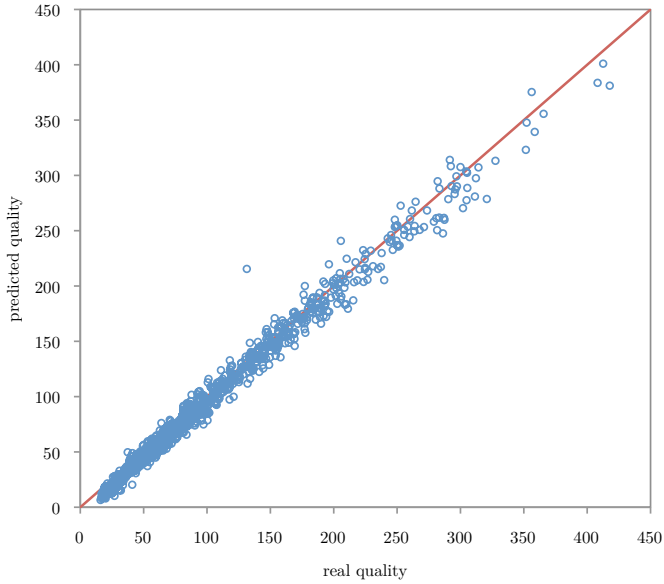
now include features related to the problem size. Such features thus appear to be important for building one general regression model for the prediction of the solution quality of all instances together. This is in contrast to the e.g. regression trees where a set of regression models are constructed for different groups of instances, depending on specific feature values.

The results of prediction models based on these reduced sets are presented in Table 4.6. It shows the correlation coefficients based on 10-fold cross-validation on the training set and when evaluated on the validation set of unseen instances. The most accurate models based on these sets are again the M5-based models and the artificial neural networks (Multilayer Perceptron). The correlation coefficients are similar to those in Tables 4.4 and 4.5. Figure 4.3 shows plots of the actual quality versus the predicted quality by the Multilayer Perceptron models built using the feature sets based on the learner-dependent feature selection process. The plots show the predictions on the validation set, (a) for `Algorithm A` and (b) for `Algorithm B` (both given the stopping criterion of 25000 schedules). The resulting prediction models are not significantly better than those based on simple correlation-based feature selection. It is thus preferable to apply the simple correlation-based feature selection approach instead of the more time-consuming linear regression-based approach.

Given the differences between the resulting feature sets of both feature selection approaches, we investigated whether certain features are more important than others, and whether some features can even be omitted. We have therefore manually removed features from the reduced sets, one by one, until the correlation coefficient of the M5P Tree model started to drop. It hereby appears that all features related to the problem size can be omitted without affecting the prediction correlation. We furthermore found that the most important features are those related to the requested resource units and the number of precedence constraints divided by the theoretical maximum number of such constraints. These seem to be features closely related to the *resource constrainedness*, the *resource strength*, and the *order strength* as defined in Section 4.2.1. We confirm this statement through the following small experiment. When we remove these features from the original set of 341 features and try to build models using all remaining features, the resulting predictions are far less accurate. The results of this experiment are presented in Table 4.7. It shows the correlation coefficients of the resulting regression tree models for the prediction of the algorithm performance given the stopping criterion of 5000 schedules. We see a significant drop in accuracy, demonstrating the crucial importance of these complexity measures for predicting the empirical hardness of the instances. While these measures were originally introduced to asses the intrinsic complexity of the problems, we have found them informative for the empirical hardness too. In the literature, several researchers concluded that none of these indicators is in



(a) `Algorithm A` (25000 schedules)



(b) `Algorithm B` (25000 schedules)

Figure 4.3: Actual versus predicted solution quality on the validation set for the Multilayer Perceptron models based on the reduced feature sets (forward linear regression-based feature selection).

		cross-validation	validation set
`Algorithm A` (5000 schedules)	Linear Regression	0.9383	0.9431
	Multilayer Perceptron	0.9876	0.9945
	Decision Table	0.9042	0.9040
	M5 Rules	0.9881	0.9902
	M5P Tree	0.9898	0.9923
`Algorithm B` (5000 schedules)	Linear Regression	0.9442	0.9468
	Multilayer Perceptron	0.9854	0.9927
	Decision Table	0.9075	0.9338
	M5 Rules	0.9869	0.9886
	M5P Tree	0.9882	0.9892
`Algorithm A` (25000 schedules)	Linear Regression	0.9412	0.9455
	Multilayer Perceptron	0.9847	0.9922
	Decision Table	0.9219	0.9332
	M5 Rules	0.9863	0.9905
	M5P Tree	0.9889	0.9902
`Algorithm B` (25000 schedules)	Linear Regression	0.9237	0.9256
	Multilayer Perceptron	0.9843	0.9902
	Decision Table	0.8837	0.9137
	M5 Rules	0.9862	0.9864
	M5P Tree	0.9889	0.9897

Table 4.6: Correlation coefficients (R) of the various models based on the reduced feature sets using a learner-dependent selection approach.

itself a clear and unambiguous complexity measure (See e.g. Herroelen and De Reyck, 1999). Our experiments confirm this statement; but more importantly, we show that, when combined, these indicators *can* lead to highly accurate predictions of the empirical hardness of multi-mode resource-constrained project scheduling instances.

		cross-validation	validation set
`Algorithm A`	M5P Tree	0.7378	0.7015
`Algorithm B`	M5P Tree	0.7820	0.7593

Table 4.7: Correlation coefficients (R) of the models without the complexity indicators of Section 4.2.1.

4.4 Applications

In this section, we will focus on applications of algorithm performance prediction models for project scheduling problems. As already mentioned in Section 2.3, performance prediction models can easily be applied in an algorithm selection framework. As we will show in the following subsection, the MMLIB benchmark library, combined with the algorithms considered in this chapter, forms an ideal setting for building such algorithm selection tools. The successful construction of such an algorithm selection tool, effectively improving over the performance of its components individually, is one of the main results of this chapter. In Section 4.4.2, we discuss a number of other possible applications of performance prediction models in the context of project scheduling.

4.4.1 An algorithm selection tool for project scheduling

When comparing the performance of both algorithms considered in this chapter, we find an interesting setting. Table 4.3 in the previous section summarises their performance on the complete MMLIB benchmark set. Given the stopping criterion of 5000 schedules, the algorithms have a competitiveness ratio of $c = 0.7957$. This competitiveness ratio is the product of a high equipotency $e = 0.8492$ and a high reach $r = 0.9370$. The equipotency indicates that both algorithms beat each other on more or less similarly sized subsets. The reach furthermore shows that these subsets are large with respect to the complete benchmark set. Indeed, the algorithms perform differently on 93.7% of the instances. Figure 4.4 shows plots of the average difference between `Algorithm A` and `Algorithm B` on the MMLIB benchmark set, given the stopping criterion of 5000 schedules, (a) for the absolute difference in makespan, and (b) for

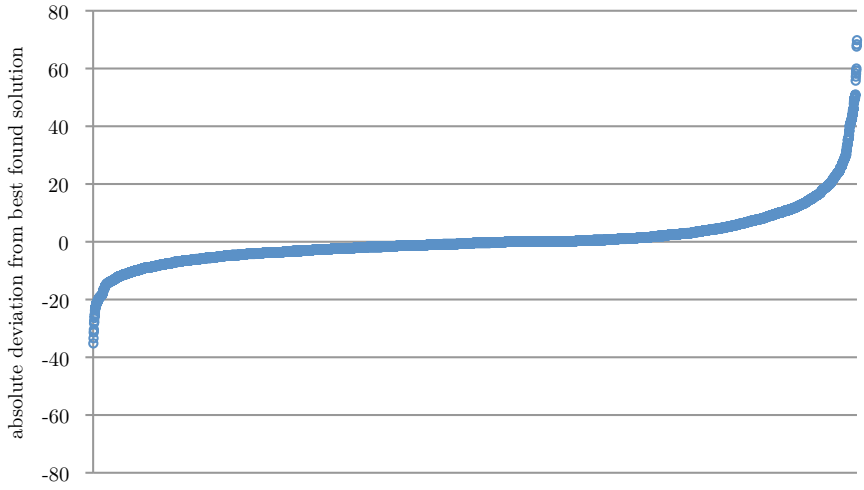
the relative difference. Both graphs are sorted on this difference.[5] We see that on about half of the instances, both algorithms perform relatively similar (i.e. the relative difference in performance is less than 5%). The difference in performance is however larger for the other half of the instances. This reflects in the potential impact, which equals $i = 2.52\%$ in this setting. Given the average best makespan in the instance set of 96.00 time units, this results in a maximal average possible improvement of 2.42 time units per instance. As will be shown below, large performance improvements can indeed be made by combining both algorithms in a selection tool based on performance prediction models.

When both algorithms are allowed to construct 25000 schedules, we get a different view. Figure 4.5 shows similar plots as Figure 4.4, but for the stopping criterion of 25000 schedules. As can be seen, `Algorithm A` has become more dominant, being the best choice on no less 89% of the instances. The equipotency has dropped to $e = 0.2354$, leading to a much lower competitiveness ratio $c = 0.2198$. `Algorithm A` seems to benefit more from longer calculation times than `Algorithm B`. The differences in performance are still relatively large, as can be seen in Figure 4.5, but the potential impact is low ($i = 0.29\%$). Given an average makespan of 90.66 time units, the maximal average performance improvement is only 0.62 time units per instance. In this setting, it will probably be considerably harder to improve over the best single-algorithm strategy than in the 5000 schedules setting.

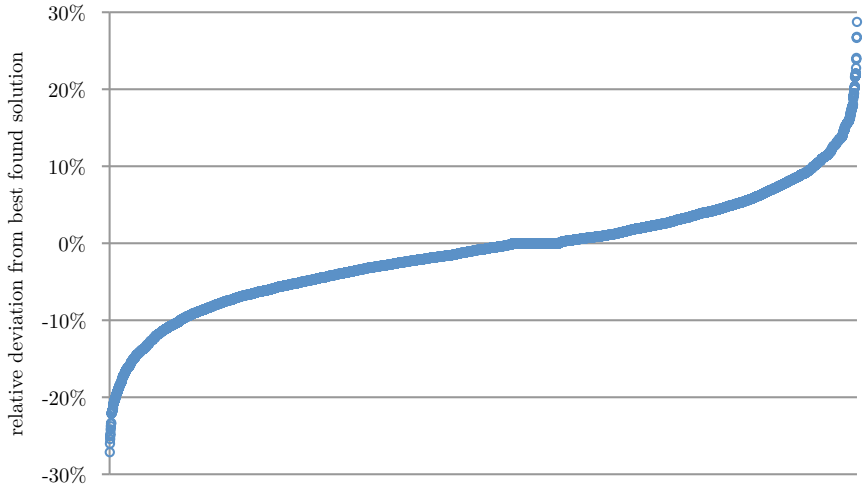
We will now investigate whether we can build successful algorithm selection tools in these two different settings (highly versus poorly competitive algorithms). It is straightforward to build such tools using the performance prediction models developed earlier in this chapter. Let `AS1` denote such an approach selecting the algorithm to run based on performance predictions for both algorithms (using the best models of the previous section). Table 4.8 summarises the overall performance of `AS1` on the validation set of unseen instances (the same one as used in the previous section), and compares the results to the case where no algorithm selection is employed (i.e. always selecting `Algorithm A` (denoted as `always-A`) and always selecting `Algorithm B` (denoted as `always-B`)) and the optimal case (where the best algorithm is always chosen, denoted as `AS*`). The results for both considered stopping criteria are displayed.

Looking at the single-algorithm strategies for the stopping criterion of 5000 schedules, we see that `always-A` is better than `always-B`, both in terms of the number of correctly classified instances and in terms of the average relative deviation from the best solution. However, as Figure 4.4 shows, the absolute

[5] In Figure 4.4(a), this difference is simply the difference in makespan; in Figure 4.4(b), the difference is measured as the relative deviation from the best found solution which is positive (negative) in case the solution of `Algorithm A` (`Algorithm B`) resulted in the longest makespan.

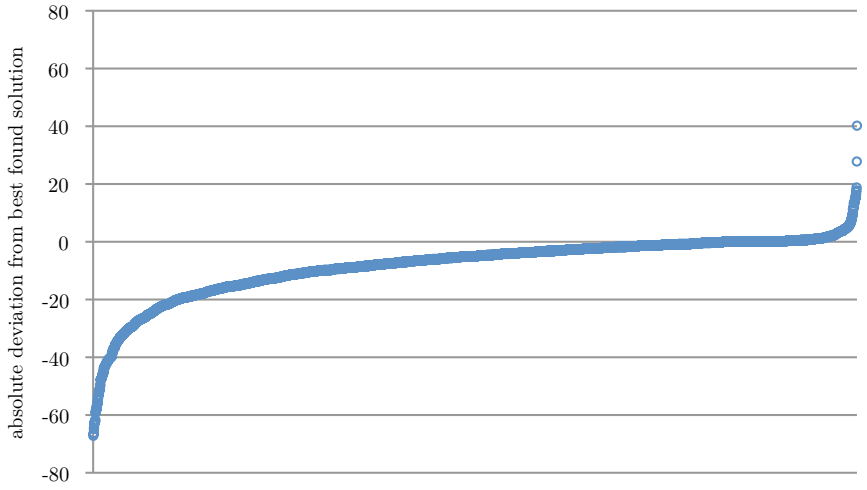


(a) absolute difference

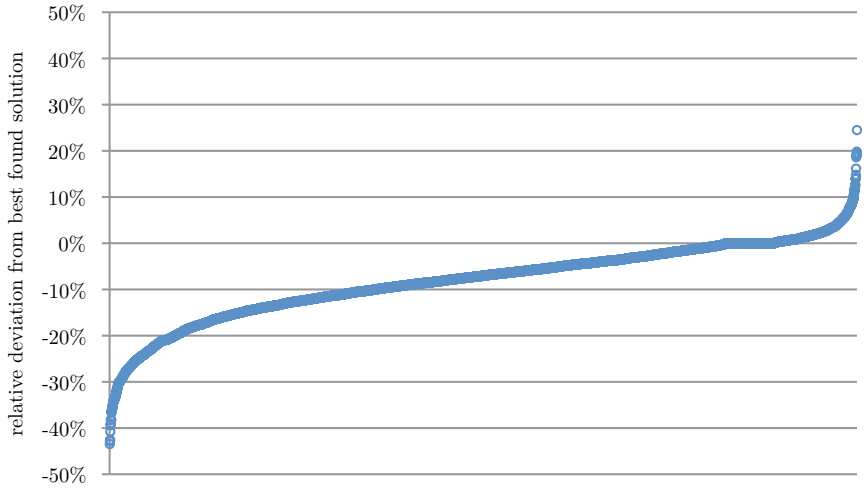


(b) relative difference

Figure 4.4: Difference in performance between `Algorithm A` and `Algorithm B` on the MMLIB benchmark set (given 5000 schedules).



(a) absolute difference



(b) relative difference

Figure 4.5: Difference in performance between `Algorithm A` and `Algorithm B` on the MMLIB benchmark set (given 25000 schedules).

		always-A	always-B	AS1	AS*
5000 schedules	% correctly classified instances	58.1	49.9	69.0	100
	avg. rel. dev. from best (%)	2.30	2.85	1.32	0
	avg. makespan	100.68	99.40	98.51	97.34
25000 schedules	% correctly classified instances	89.6	18.0	80.1	100
	avg. rel. dev. from best (%)	0.36	8.01	1.04	0
	avg. makespan	92.25	99.18	92.66	91.90

Table 4.8: Performance of the AS1 algorithm selection strategy on the validation set.

difference in performance between both algorithms is larger for those instances where Algorithm B is better. When we thus look at the average makespan on the validation set, we see that always-B is actually better than always-A. This shows that it is not sufficient to just check the number of correct decisions. It is important to look at the overall performance of the selection strategy as well. For the stopping criterion of 5000 schedules (where the algorithms have a competitiveness ratio of $c = 0.7957$), Table 4.8 shows that AS1 is able to effectively improve on the results of the best single-algorithm strategy in terms of all three considered criteria.

For the stopping criterion of 25000 schedules on the other hand, AS1 can not improve over the results of always-A. In this setting, the algorithms are far less competitive ($c = 0.2198$), due to the increased dominance of Algorithm A over Algorithm B. As argued in Section 2.3.1, this low competitiveness ratio indicates that an algorithm selection tool has to be considerably more accurate to be able to beat the best single-algorithm strategy. AS1 is in this setting unable to achieve such a level of accuracy. This might be a consequence of the fact that AS1 relies on two (possibly erroneous) predictions, which have to be compared. In the case study on nurse rostering in Chapter 3, we ran into similar problems. The solution was found in a different approach towards algorithm selection tools.

Instead of straightforwardly applying the models predicting the performance of one algorithm, it might be more useful to employ classification models predicting the best algorithm choice directly. Let AS2 denote the algorithm selection tools based on such classification models. Table 4.9 shows the performance of AS2 for three of the best performing models, evaluated on the validation set of unseen instances. AS2a denotes the algorithm selection tool using a REP-Tree

model, which is a decision tree that is pruned using a reduced-error-pruning scheme. `AS2b` denotes the tool using a DecisionTable model (consisting of 240 (219) rules for the 5000 (25000) schedules case). `AS2c` denotes the tool using a RandomForest model. This is a collection of (in our case 500) decision trees for which the feature sets contain 30 randomly selected features. The RandomForest outputs the class that was the most predicted by the random trees. We see that for both considered stopping criteria, the RandomForest model performs best.

For the stopping criterion of 5000 schedules, we see that `AS2` significantly improves over `AS1`. Comparing Tables 4.8 and 4.9, we see that `AS2` further reduces the relative deviation from the best found solution from 1.32% to 0.51%. The average makespan is very close to the optimal average makespan. While the `AS1` approach led to good results for highly competitive algorithms, the `AS2` approach even further improves on these results, leading to near-optimal performance. For the stopping criterion of 25000 schedules, where the algorithms are only poorly competitive, we see that `AS2` is able to effectively improve over the results of `always-A`. The relative deviation from the best found solution is reduced by half (from 0.36% to 0.18%). The `AS2` strategy is thus able to achieve much higher accuracy levels than the `AS1` approach. The resulting prediction accuracy is even sufficient to achieve improvements in the 25000 schedules setting, where the algorithms are only poorly competitive.

In an additional experiment, we applied feature selection techniques, investigating whether smaller feature sets could also lead to similar results. We have therefore applied forward, backward and bi-directional correlation-based feature selection procedures for both stopping criteria. The resulting feature sets are given in Tables D.13 – D.17 in Appendix D. These sets again contain a considerable amount of features related to the complexity measures of Section 4.2.1. Let `AS2'` denote the algorithm selection tools based on these reduced feature sets. Table 4.10 summarises the performance of these tools, evaluated on the validation set of unseen instances. `AS2a'`, `AS2b'` and `AS2c'` correspond to selection tools based on the same machine learning techniques as `AS2a`, `AS2b` and `AS2c`. `AS2d'` denotes a tool using artificial neural networks (i.e. multilayer perceptrons). When comparing the results of `AS2'` to those of `AS2`, we can conclude that the feature selection process did not significantly decrease the quality of the results. In some cases, there is even a slight improvement. Generally, the `AS2c'` approach (using random forest models) based on backward correlation-based feature selection leads to the best results.

		AS2a	AS2b	AS2c	AS*
5000 schedules	% correctly classified instances	80.2	77.1	83.2	100
	avg. rel. dev. from best (%)	0.67	0.79	0.51	0
	avg. makespan	97.93	98.05	97.74	97.34
25000 schedules	% correctly classified instances	92.2	91.4	93.3	100
	avg. rel. dev. from best (%)	0.24	0.27	0.18	0
	avg. makespan	92.05	92.08	92.02	91.90

Table 4.9: Performance of the AS2 algorithm selection strategy on the validation set.

			forward	backward	bi-directional
5000 schedules	AS2a'	% correctly classified instances	78.4	78.4	79.4
		avg. rel. dev. from best (%)	0.73	0.75	0.69
		avg. makespan	97.98	97.98	97.91
	AS2b'	% correctly classified instances	77.4	77.4	77.4
		avg. rel. dev. from best (%)	0.86	0.86	0.84
		avg. makespan	98.12	98.12	98.10
	AS2c'	% correctly classified instances	81.5	81.9	81.7
		avg. rel. dev. from best (%)	0.58	0.59	0.57
		avg. makespan	97.84	97.83	97.85
	AS2d'	% correctly classified instances	77.9	78.3	79.5
		avg. rel. dev. from best (%)	0.79	0.71	0.67
		avg. makespan	97.97	97.91	97.87
25000 schedules	AS2a'	% correctly classified instances	91.7	92.6	91.7
		avg. rel. dev. from best (%)	0.26	0.17	0.26
		avg. makespan	92.07	92.03	92.07
	AS2b'	% correctly classified instances	91.6	91.8	91.6
		avg. rel. dev. from best (%)	0.27	0.22	0.27
		avg. makespan	92.06	92.05	92.06
	AS2c'	% correctly classified instances	93.4	93.5	93.4
		avg. rel. dev. from best (%)	0.18	0.18	0.18
		avg. makespan	92.03	92.02	92.03
	AS2d'	% correctly classified instances	91.2	90.3	91.2
		avg. rel. dev. from best (%)	0.30	0.31	0.30
		avg. makespan	92.09	92.12	92.09

Table 4.10: Performance of the AS2' algorithm selection strategy on the validation set, based on different correlation-based feature selection techniques.

4.4.2 Other applications

Apart from algorithm selection, other possible application domains exist.

During the planning phase, the project management team could employ performance prediction models as a supporting tool. Sometimes, projects can be divided over different facilities of a production plant. In order to decide on a good subdivision, performance prediction models could be used to get quick estimates of the resulting qualities (e.g. makespans) for different scenarios.

Another possible application could be found in capacity planning, where strategic decisions have to be made on how to expand the current infrastructure or supply demands. Many different scenarios can be quickly evaluated using performance prediction models.

4.5 Conclusions

In this chapter, we have demonstrated the feasibility and usefulness of performance prediction models for the multi-mode resource-constrained project scheduling problem.

We started by reviewing a number of important complexity indicators proposed in the literature. These complexity indicators were originally defined for the single-mode version of the project scheduling problem. With the aim of building accurate performance prediction models, we proposed an extensive feature set including (transformations of) these indicators. Looking at generalised project scheduling problems (in our case multi-mode resource-constrained project scheduling problems), this implied that the complexity indicators would have to be generalised as well. In many cases, this meant that one indicator value was transformed into a vector of values. The resulting feature set is applicable to other generalisations of the project scheduling problem too. This feature set (as described in Appedix C) is part of the main results of this chapter.

As a basis for the experimental study, we employed the MMLIB benchmark library. This library contains larger and more diverse instances than the older, but more widely used PSPLIB benchmark set. Using the MMLIB set had the further advantage that the considered algorithms were competitive, enabling us to also investigate a practical application based on empirical hardness models.

We investigated performance prediction models for two inherently different state-of-the-art algorithms: a tabu search algorithm and a (hybrid) genetic algorithm. We hereby experimented with two different stopping criteria, leading

to a number of interesting settings. The performance prediction models are based on our newly defined feature set, and are shown to be highly accurate for both considered stopping criteria. The most accurate models were those based on regression trees and decision rules. We furthermore showed that the application of feature selection techniques does not deteriorate the accuracy of the predictions.

Based on the developed performance prediction models, we investigated the relative importance of the (variants of the) complexity indicators presented in the literature. We have demonstrated that these indicators are of crucial importance for the success of the prediction models. When the features related to these indicators are left out, none of the considered machine learning techniques are able to produce accurate performance predictions.

Based on our success in building accurate performance prediction models, we investigated a practical application of such models in the form of automatic algorithm selection tools. The two considered stopping criteria provided us with two different settings: a highly competitive one and a poorly competitive one.

We have built automatic algorithm selection tools in two different ways. The first approach (`AS1`) simply compares performance predictions for the set of algorithms and decides which algorithm to run based on these predictions. This approach leads to good results in case the algorithms are highly competitive (i.e. in the 5000 schedules case). However, for poorly competitive algorithms (i.e. in the 25000 schedules case), this approach was not sufficiently accurate to outperform the best single-algorithm strategy.

The second approach (`AS2`) outperforms the first approach. Here, we built only one classifier predicting a class (i.e. the algorithm that will probably perform best). These predictions are also based on the developed feature set. Since only one prediction is made instead of two, we reduce the probability of making wrong selections. We found that the best performing technique is based on building random forests consisting of decision trees. As for the construction of empirical hardness models, the application of feature selection procedures did not decrease the accuracy of the resulting models. We have shown that this approach leads to near-optimal results on a validation set of unseen MMLIB benchmark instances. It is even sufficiently accurate for achieving performance improvements in case of poorly competitive algorithms.

In this chapter, we have thus proven the practical usefulness of performance predictions for project scheduling and we have successfully built an automatic algorithm selection tool reaching near-optimal results and outperforming any of its components individually. This is the case for both highly and poorly competitive algorithms.

Chapter 5

Practical considerations

In this dissertation (in Section 2.2), we have proposed a framework allowing for the construction of empirical hardness models for practical combinatorial optimisation problems. This framework is formulated as a five-step procedure, describing the necessary ingredients at a high level. In subsequent chapters, we have applied this framework in two practical settings. In a sense, this can be seen as a simple application of a given framework. However, we found this task to be non-trivial. The high-level description of the framework does not specify *how* its ingredients should be instantiated in a practical setting. These instantiations are furthermore crucially important for the success of the strategy. In this chapter, we will address this issue. We will discuss how such specific instantiations can be built, based on our experience gained throughout this doctoral project. We will formulate this through a number of good practices, interweaved with examples from our own studies.

The following sections (5.1−5.5) each discuss one of the steps of the framework. We present in-depth discussions on how we have built the necessary ingredients and formulate advice towards doing so in other contexts. Afterwards, in Section 5.6, we conclude this chapter with a short summary.

5.1 Instance distribution (Step 1)

We have already stressed the importance of the instance distribution at several places throughout this dissertation. The limits of the instance distribution are key to the successful application of the proposed strategy in any setting. The

resulting models are only expected to be valid within these bounds. Sometimes, however, good performance can be generalised outside this distribution. Models built using a training set of instances inevitably focus on this instance set. It is thus important to have a set of instances similar to the instances on which the learnt models should be applied. Determining the properties of these latter instances can be a difficult task. Sometimes, the specifics of the instance distribution are not known in advance. Such distributions might even change over time. It is crucial to have a representative instance set for the target domain.

This can be achieved through two possible approaches. A large dataset containing instances known to be similar to the target instances may be available. In this case, there is no explicit description or understanding of the underlying instance distribution necessary. Such a dataset might be found through the inspection of log files, or could be based on benchmark data. An example of the latter is found in our experiment on project scheduling in Chapter 4. When no such large dataset is available, it is necessary to explicitly state the properties of the target instances. This requires an in-depth investigation of the application domain, or thorough discussion with domain experts (e.g. the end users). It is important to identify the key parameters of the problem and to determine appropriate ranges and/or distributions for their values. This allows for the creation of an instance generator, which can be used in Step 4 of the framework to generate a training set. An example of this approach can be found in our experiment on nurse rostering in Chapter 3. Given the problem definition, and a number of target instances from the 2010 INRC, we determined the properties of the instances of interest. We set ranges for the different parameters (e.g. the number of nurses, their requests, appropriate working demands) and determined relevant contract types.

5.2 Algorithm set (Step 2)

In Step 2 of the procedure, a set of algorithms is to be selected. In case it is known for which algorithms the predictions need to be made, instantiating this step is easy: just select those algorithms and decide on the performance criterion to be predicted.

In the case an algorithm selection tool is to be built, it might be less clear which algorithms should be included. Aiming at formalising this task, we proposed a theoretical framework allowing for the characterisation of a set of algorithms in terms of their competitiveness and potential impact in a portfolio solver in Section 2.3.1. The notion of competitiveness gives an indication of

how accurate a selection tool will have to be in order to be able to improve over the best single-algorithm strategy. Highly competitive algorithms can benefit from a moderately accurate selection strategy, while poorly competitive algorithms require extremely accurate selection tools. In its current formulation, the framework only considers sets of two algorithms. In ongoing work, we are investigating how these concepts can be generalised to larger algorithm sets.

5.3 Feature selection (Step 3)

The performance prediction models in this dissertation aim at predicting the empirical hardness of instances being solved by particular algorithms. Such models make their predictions based on an aggregate view of the instances. This view is based on a feature set, characterising the instances by means of a vector of feature values. As we have argued in this dissertation, designing such a set is not straightforward. What one is looking for, are structural properties of the instances relating to algorithm performance. There is no universal way of finding such properties, although there are some common practices.

In this section, we do not want to present an exhaustive list of possible features. Instead, we want to explain how we got to the feature sets presented in Appendices A and C. We will hereby provide a number of useful ways of thinking about features.

Several options exist towards characterising problem instances by means of a vector of feature values. Sometimes, such feature sets already exist. In that case, they can easily be employed, without the need for a deeper understanding of the problem domain. In most cases however, such feature sets do not yet exist. Two possible approaches are: (1) designing a new feature set (as done for both nurse rostering and project scheduling) or (2) translating your problem to one for which a feature set already exists (as done in the proof-of-concept experiment on nurse rostering).

5.3.1 Designing a new feature set

In this dissertation, we designed two feature sets for two distinct combinatorial optimisation problems. We began with simple features, which could easily be calculated by iterating over the concepts in the problem description. Depending on the problem description, this may lead to a large number of feature values. We found in both settings, that these sets allowed for highly accurate performance prediction models, even without incorporating more complex (and

time-consuming) features. Our experience indicates that complex features are not strictly necessary for the resulting models to be highly accurate. Consequently, we will only discuss how to identify simple features based on the problem description.

When designing a new feature set for a given problem domain, it is important to identify the key concepts in the problem description. This description gives an overview of all elements present in the domain. It is useful to think about a variety of features related to size, constraint values, matrix representations and graph representations. Each of the following subsections discusses one of the categories mentioned above. Please note however, that some features could be categorised in more than one category. It is not our aim to present a strict classification. Instead, we want to provide a discussion on how relevant features can be identified. It is important to cover a wide array of possible feature values. In many cases, some values will be irrelevant in the context of the target instances. Such irrelevant features will eventually be removed in Step 4.

Size-related features

Many concepts present in a description can easily be counted, and these values can straightforwardly be included as size-related features. Examples for the nurse rostering problem are the total number of nurses, shift types, skill types, days, etc. For project scheduling, we can think of the number of projects, (non-) renewable resource types, resource units, precedence relations, etc. In essence, for each concept present in the description, there is generally a feature to be identified relating to the size of this concept.

It may furthermore be useful to express certain relationships or ratios between size-related features. When thinking about project scheduling problems, examples of such features are the number of precedence relations divided by the number of activities, the number of resource types divided by the number of activities, etc. Sometimes, these relationships are a bit more complex, and actually more closely related to the constraint values. Examples are the skill types in the nurse rostering problem. Instead of simply including a ratio of the number of skill types over the number of nurses, it may be more useful to include statistics on the distribution of skill types over the nurses (and vice versa). Such features however, are more related to the problem structure and will be discussed in the following subsection.

It is sometimes possible to express certain size-related features in relation to a theoretical maximum. For example, the number of precedence constraints divided by the theoretical maximum number of such constraints gives an indication of how constrained a project scheduling problem is.

Constraint-related features

Not only the number of concepts present in the description is important, also the properties of these concepts can be used as a source for features.

Simple constraint values can straightforwardly be included in the feature set. In the proof-of-concept nurse rostering study e.g., we included the contract's constraint values as features. In most cases, such constraint values are varied over certain concepts. In that case, aggregate statistics (e.g. the mean, minimum, maximum, minimum-over-maximum ratio, variation and standard deviation) on the distribution of these values can be included. In the extended feature set for nurse rostering problems, we included such statistics on the constraint values. As these constraints vary over the set of contracts, we included the aggregate values over the contracts (e.g. the mean maximum number of shifts to be worked over the four contract types). These contracts are furthermore related to a number of nurses. We could also look at the distribution of the constraint values over the nurses. Consequently, we included aggregate statistics over the nurses. Similar ideas apply to the project scheduling feature set. In this problem, e.g. resource requirements are varied over the modes of each activity. Statistics on this distribution can be calculated for each activity (call these lower-level statistics). On a higher level, various statistics of these lower-level statistics can then be calculated over all activities. We included the mean, minimum, maximum, minimum-over-maximum ratio, variation and standard deviation of the mean number of resource types required by the different modes of an activity. This can also be done for the requested resource units, and for the durations of the modes. This process may lead to a large number of features. Other examples in the project scheduling case include statistics on the capacity of different resource types. Such features can be included for all resource types, and for both renewable and non-renewable types separately.

The relationships between different concepts give rise to relevant features. If certain concepts are 'owned' by other concepts, statistics on their distribution can be included in the feature set. We briefly mentioned such features in the previous subsection. In the context of nurse rostering problems, we included various statistics (the mean, minimum, maximum, minimum-over-maximum ratio, variation and standard deviation) of the number of nurses per contract, the number skill types per nurse, the number of requests per nurse, the number of requests per day, etc. The inverse of such relationships may lead to relevant features. An example is the relationship between the nurses and their skill types. We can consider statistics on the number of skill types per nurse, but also on the number of nurses per skill type.

Similar to the size-related features, there may exist interesting relationships

between the considered constraint-related features. Such relationships could be relatively simple, as in the ratio of the minimum over the maximum value of certain features over a set of concepts. More elaborate relationships may be expressed through ratios representing the requirements versus the availabilities of certain concepts. Such ratios express a certain tightness of the constraints; or when inversely interpreted, a certain freedom in solving the problem. As we have shown in this dissertation, features related to such relationships are among the most important ones. A useful way of identifying such features is to look for properties relating to an availability (or capacity) of resources and putting them in relation to the constraints specifying the corresponding demand. Additionally, whenever certain constraints pose a minimum and a maximum value on certain properties, their minimum-to-maximum ratio can be considered as an indicator of the degree of freedom in finding a satisfying assignment. It is straightforward to understand that such a ratio gives an indication of how much freedom there is in solving the instances. The closer this ratio is to one, the tighter the bounds are in which a solver can operate. In the feature set for nurse rostering problems, we included such ratios representing a degree of freedom in solving the instances. An example is the minimum-over-maximum ratio of certain constraints limiting the length of series of working (or free) days. We also included features concerning the relationship between the requirements and the availabilities. An example is the `ratioAvailabilityOverCoverage` feature, representing total number of shifts that can be worked by the staff divided by the total number of shifts that need to be worked. This particular feature has been shown to be extremely important for the prediction of algorithm performance. Other examples include the maximum number of assignments over the minimum number of consecutive working days and the minimum number of consecutive free days over the minimum number of consecutive working days. In the project scheduling case (and in general in any case where capacity constraints apply), ratios expressing the requirements to their respective availabilities are good candidate features. In our case, each activity has a number of possible execution modes, each with separate requirements. We considered a number of statistics (e.g. mean, minimum, maximum, etc.) on the requirements and have put these in relation to the availabilities.

features based on matrix representations

In some problem descriptions, certain constraint values are, or can be, represented by matrices. The properties of these matrices may be considered as features. Examples are features related to the diagonal dominance, and the density or sparsity of the matrix (i.e. the fraction of zero or non-zero elements). If the matrix furthermore contains distances, it may also be useful to include the degree to which the triangle inequality is satisfied.

In the case studies presented in this dissertation, we did not include such features, as the problem descriptions did not lead to straightforward or meaningful matrix representations.

features based on graph representations

Many problem descriptions straightforwardly lead to one or more graph representations. In general, as soon as two (not necessarily different) concepts in a problem domain are related, a graph can be constructed which visually represents this relationship. Graph representations may lead to an additional set of features. Evident examples are the number of nodes and edges in the graph, and various statistics on the degree of the nodes. It may furthermore be interesting to put the number of edges in relation to the theoretical maximum number of edges in such a graph.

Graphs can be built up based on a variety of concepts in the problem description. Sometimes, the relationships may be one-way only, in which case the graph is a directed graph. Such graphs can furthermore be extended with dummy source and sink nodes. Relationships between different types of concepts may be represented by graphs containing different types of nodes. The resulting graphs may be furthermore be bi-, tri-, or multipartite. In such cases, degree statistics for each type of node may be included.

Examples can be found in the SAT feature set described in Section 2.2.1. This set includes features related to three types of graphs. The clause graph contains a node for each clause in the formula and connects nodes when they share a variable. The variable graph contains a node for each variable and has edges whenever two variables both appear in a clause. The clause-variable graph is a bipartite graph a clause node for each clause and a variable node for each variable. Whenever a variable is contained in a clause, there is a link between the corresponding nodes.

In the nurse rostering feature set, we did not consider strict graph representations. We could have considered e.g. a nurse graph containing a node for each nurse and edges whenever the corresponding nurses work according to the same contract. This graph however, would lead to features which were already included as size- or constraint-related features (i.e. statistics on the number of nurses working according to each contract). In the feature set for project scheduling, we could have considered the precedence graph, representing all precedence relations between activities. But as for the nurse rostering case, statistics on the node degrees were already included as size- or constraint-related features. Moreover, the feature sets developed based on the previously discussed principles were already sufficient for achieving highly accurate performance predictions. We

therefore did not consider additional graph representations in the case studies presented in this dissertation.

5.3.2 Translating the problem

The second option towards characterising problem instances by means of a feature vector is through a translation into another problem for which a feature set already exists. We investigated such an approach in the case study on nurse rostering problems. In collaborative work, we developed a translation scheme allowing to represent nurse rostering instances as propositional satisfiability problems. This translation scheme is not part of this dissertation, and its details are thus left out of this discussion. We refer to Haspeslagh (2012) for a description of the scheme. Nevertheless, we can mention that developing this translation scheme required a significant insight in both nurse rostering and SAT problems. It furthermore required a detailed understanding of the numberings proposed by Burke et al. (2001), in order to represent the different constraints by means of boolean variables. This approach thus required a considerable effort.

One could argue that employing such a translation scheme eliminates the need to design a feature set for the specific problem at hand. This is true to a certain extent, in the case this translation scheme and the appropriate feature set are already available. However, *developing* such a translation scheme requires in our view more knowledge and effort than designing a novel feature set. We furthermore demonstrated that the SAT feature set did not lead to better results than using only the NRP feature set (which was even very limited in the proof-of-concept experiment). We would thus rather argue that it is easier to develop a new feature set for a given problem, than to design a translation scheme and use an existing feature set.

5.4 Data generation (Step 4)

For any prediction model based on machine learning techniques to be successful, it is necessary that sufficient and relevant data is available for training. As we have mentioned above (while discussing Step 1), it is crucially important that the data on which the model is trained is sufficiently similar to the data on which predictions are to be made. The definition of the problem instance distribution should thus not be underestimated. We have discussed two possible approaches towards identifying this instance distribution. This can be done either implicitly, through the availability of a large dataset; or explicitly, by

stating the key parameters and their appropriate value ranges (which then allows for the creation of an instance generator). In both cases, this leads to a large enough dataset containing instances similar to the target instances.

Given this dataset, it is then necessary to determine the meta-data needed for building the performance prediction models. This includes calculating all feature values for all instances, and determining the performance criteria for the selected algorithms. The latter implies running all algorithms on all instances, which is a time-consuming task. In this dissertation, we have therefore employed a massive parallel computing infrastructure. Through this approach, we have significantly reduced the total calculation time.

Once all meta-data is available (i.e. performances of all algorithms and feature values for all instances), it is advised to perform a preprocessing step. We have stressed the importance of having a qualitative feature set. It is better to have too many features than to have missed some important ones. The result may thus be that the instances are represented by very large feature vectors. These feature vectors possibly contain irrelevant features, which could complicate or slow down the machine learning process. We performed a simple data cleaning step before the actual model construction in Step 5. This step includes the removal of uni-valued and perfectly correlated features.

5.5 Performance prediction models (Step 5)

A large variety of techniques exist for the construction of performance prediction models. Throughout this dissertation, we have experimented with many such techniques.

In theory, finding the best prediction models can be achieved by trying out all possible techniques. In practice however, this leads to a huge waste of time, as many techniques require a non-negligible construction time (especially in the case many features are available) and many resulting models would only lead to poor predictions. We have found the WEKA software tool to be particularly useful for such experimentations. It is an easy-to-use toolbox, including state-of-the-art implementations of many commonly used machine learning techniques. It is based on an easy-to-understand data format and is furthermore open-source software, being freely available under the GNU General Public License.[1]

Throughout our experiments, we have found certain techniques to be performing generally better than many other, regardless of the specific setting. In what follows, we will discuss our findings in separate subsections, according to the

[1] More information on http://www.gnu.org/licenses/gpl.html

performance criterion that is to be predicted. We hereby distinguish running time predictions, solution quality predictions and algorithm choice predictions. Afterwards, we discuss feature selection techniques, allowing the construction of much smaller models achieving similar accuracy levels.

5.5.1 Running time prediction

We have considered running time predictions exclusively in the proof-of-concept study on nurse rostering. This criterion is only relevant when considering complete search methods aiming at finding an optimal solution. When considering practical combinatorial optimisation problems being solved by metaheuristic algorithms (which is the topic of this dissertation), this criterion is far less interesting.

While other researchers achieved accurate prediction models for the running time of complete algorithms, we were not able to do so. A few examples of such success stories are for the winner determination problem within combinatorial auctions (Leyton-Brown et al., 2006) and the propositional satisfiability problem (Nudelman et al., 2004). These authors have mainly applied statistical linear regression or ridge regression techniques and have reported good results for the prediction of the logarithm of the running time. Based on these results, we could thus advise to apply regression techniques on a logarithmic transformation of the running times.

We did investigate other ways of finding useful results in such settings. An indication of whether the running time is above or below some threshold might suffice for certain practical purposes. In the proof-of-concept study on nurse rostering, we did find accurate models predicting whether the running time would be long or short. We hereby found the random forests technique to be producing the most accurate results.

5.5.2 Solution quality prediction

The main body of this dissertation focused on the prediction of solution qualities. In the proof-of-concept study on nurse rostering, we looked at both optimal and approximate solutions, depending on the type of algorithm that was used. In the other experiments, we focused mainly on the solutions obtained by metaheuristic algorithms. The quality of the resulting solutions is represented by a numeric value and is generally dependent on the problem description. For the nurse rostering problems, we considered the value of the objective function, which is a weighted sum of the constraint violations. For project scheduling, we

defined the quality of a solution as the makespan, i.e. the finishing time of the last scheduled activity.

With respect to solution quality predictions, we have found that the most promising techniques are based on regression trees, decision tables and artificial neural networks. The regression trees (MP5 trees) were hereby leading to the most accurate results. Note that artificial neural networks require more training time, and are hence not applicable when large feature sets are considered.

5.5.3 Algorithm choice prediction

An important application of algorithm performance prediction models is found in automatic algorithm selection tools. In this dissertation, we have built such tools for both nurse rostering and project scheduling problems. While we have shown that performance prediction models can directly be applied in such a tool; this was only successful for highly competitive algorithms.

Alternatively, we investigated whether predicting the algorithm choice directly would lead to better results. This was indeed the case. The resulting models reached the necessary level of accuracy to improve even in the case of poorly competitive algorithms. In both case studies presented in this dissertation, we have found that the models based on random forests provided the most accurate predictions. We therefore advise to apply this classification technique when constructing algorithm selection tools, instead of straightforwardly applying separate performance prediction models for each algorithm. This finding is consistent with the results of recent work by Hutter et al. (2014).

5.5.4 Building smaller models

After cleaning the data in Step 4 (i.e. removing irrelevant features), the feature vectors may still be quite large. As we have shown in both case studies in this dissertation, feature selection techniques can significantly reduce these vectors without affecting the prediction accuracy of the resulting models. Such feature reductions furthermore reduce the risk of over-fitting, and raise the potential for interpreting the resulting models.

Throughout this dissertation, we have applied several different methods for reducing the feature set. For the nurse rostering problem, we found that the correlation-based feature selection approach did not always result in a qualitative feature set. In the project scheduling case on the other hand, the backward correlation-based feature selection approach led in all cases to qualitative feature sets. As an alternative, a learner-dependent technique based

on linear regression models was applied in the nurse rostering case, leading to good feature sets allowing for prediction models of similar accuracy as when the complete feature set was used. The main disadvantage of this technique is that it is computationally more expensive than a rather 'simple' correlation-based approach.

We would thus advise to take an iterative approach while constructing performance prediction models. In a first iteration, prediction models should be built using all relevant features. This sets a baseline to which prediction models using smaller feature sets can be compared. Then, correlation-based feature selection techniques should be applied. If the resulting models are similarly accurate, the process can be stopped and the resulting models can be considered as being the final outcome. In case the resulting models are less accurate, more elaborate feature selection techniques like linear regression-based feature selection could be applied.

Note that when comparing the resulting models, it is important to evaluate them based on a validation set of unseen instances. If such a set is not available, 10-fold cross-validation could be used instead.

5.6 Conclusions

In this chapter, we have formulated a number of good practices for building performance prediction tools. These are based on our experience gained throughout this doctoral project. We have applied the five-step procedure for constructing empirical hardness models, proposed in Chapter 2, to two different case studies. As we have mentioned in the beginning, it is important that all its ingredients are well-instantiated. Because this task is not straightforward, we dedicated this chapter to discuss *how* such practical instantiations can be built.

We have structured this chapter along the lines of the proposed framework. We stressed the importance of the instance distribution, as the resulting prediction models are only expected to be valid within this distribution. The considered instances must furthermore be characterised by a number of informative features. Constructing such a feature set must be done with care. We have shown how this can be done in two prototypical cases and have given numerous examples. Regarding Step 4, we have pointed at high performance computing infrastructures for speeding up the time-consuming task of generating the training data. Once the training meta-data is available, the actual construction of prediction models can start. We discussed the machine learning techniques resulting in the most accurate models. We did this for the different performance criteria separately. We pointed at feature selection techniques leading to

significantly smaller models with similar accuracy. Such smaller prediction models reduce the risk of over-fitting and have a higher potential for experts to understand the relationship between feature values and algorithm performance.

Chapter 6

Conclusions and further work

In this concluding chapter, we will summarise the main results of the doctoral project described in this dissertation. In Section 6.1, we will reformulate the research questions and discuss how we have successfully answered them through the different chapters of this dissertation, focusing on the main contributions. In Section 6.2, we will point at a number of directions for further research.

6.1 Summary and contributions

In this dissertation, we investigated algorithm performance predictions for practical combinatorial optimisation problems. In Chapter 1, we presented a short introduction, discussing the challenges and revealing the main research questions at the core of the doctoral project positioned at the intersection of artificial intelligence and operational research.

The project is based on an existing framework for the prediction of running times of complete algorithms solving decision or optimisation problems. This framework had a number of limitations and hence, the main research question quickly emerged:

> *How can this framework for running time predictions of complete search methods be adapted to handle more practical settings where optimisation problems are solved using metaheuristic methods?*

We have answered this question in Chapter 2. The aim of this chapter was to

build a relevant context for the project. This goal was achieved through the combining a literature overview with the introduction of novel ideas.

The main contribution (and the answer to the main research question) was the formulation of a simplified and generalised strategy for the construction of performance prediction models for practical combinatorial optimisation problems. With respect to the original formulation, this strategy allows for a much broader spectrum of potential applications. The most important generalisation is the inclusion of the selection of a performance criterion in Step 2. This is based on our definition of *empirical hardness*, which allows it to be measured by *any* performance criterion.

Due to its generality, the framework is formulated at a high level. It does not include any specific details on how it is to be applied in a practical setting. As a consequence, the main research question gave rise to two more specific questions related to two prototypical problem domains:

> *How can the framework be applied to the nurse rostering problem?*

and

> *How can the framework be applied to the multi-mode resource-constrained project scheduling problem?*

The answers to these questions are presented in detail in Chapters 3 and 4. These chapters discuss the extensive experimental investigation towards performance predictions in the respective problem domains. In these chapters, we have successfully built predictive systems for state-of-the-art algorithms in a number of different settings. This required designing extensive feature sets for the characterisation of instances in both problem domains. These feature sets are among the important contributions, as such sets were not yet available in the literature.

An important application of such predictive systems is found in automatic algorithm selection tools. In Chapter 2, we introduced a theoretical framework for the characterisation of a set of algorithms to consider in such an application. This framework is based on the concepts of *competitiveness* and *potential impact*. For both problem domains, we showed that the simple application of performance prediction models in such a tool was not successful for poorly competitive algorithms. This approach is based on comparing the predicted individual performances of algorithms and is not sufficiently accurate in this setting. An alternative approach, directly predicting an algorithm choice based on the feature values, did reach the necessary accuracy levels. In both case studies,

we built an algorithm selection tool consisting of state-of-the-art algorithms, outperforming any of its components individually and reaching near-optimal results. These applications are also among the main contributions of Chapters 3 and 4.

Constructing specific instantiations of the ingredients of the framework, leading to its successful application in two different settings, was not trivial. We anticipated this and therefore formulated a final research question:

> *How should the ingredients of the framework be instantiated in any practical setting?*

Answering this question in a general way is inherently difficult. There are so many different settings, making it hard to formulate strict rules or guidelines. Instead, we approached this question in Chapter 5 by providing a thorough discussion on the important aspects of the framework, interweaved with examples from our own experience. For each step of the strategy, we formulated a number of good practices and provided concrete examples.

6.2 Future directions

As with any doctoral research project, a number of choices had to be made limiting the ultimate scope of the presented work. Consequently, a number of questions remain, and a range of directions for further research are open to the communities. We discuss a number of these possibilities but stress that this list is by no means exhaustive.

Other algorithms and instance distributions

A straightforward extension of the work presented in this dissertation is in adding more algorithms to both case studies. Especially in the case of algorithm selection, this could lead to further performance improvements. Note that such algorithms do not necessarily have to be outperforming others on many instances. Even algorithms working well only for a limited set of instances can be valuable elements of an algorithm selection tool.

Investigating other instance distributions can also lead to a number of insightful results:

- It is interesting to investigate whether the same features are still important for other instance distributions (or even other algorithms).

- It could be checked whether the prediction models have any general significance, i.e. whether they can be straightforwardly used to predict the performance on other instance distributions.

  It is interesting to investigate to what extent this could be done by iteratively adapting the instance distribution until the models start to loose their accuracy.

Other problem domains

Evidently, the proposed framework could be applied to any other problem domain. The discussion in Chapter 5 should facilitate this process.

Building practical applications based on performance predictions

In this dissertation, we have applied the algorithm performance prediction models in automatic algorithm selection tools. As mentioned above, these applications could be extended by including other algorithms.

Performance prediction models could also be applied in other applications. Such models are particularly useful when quick performance estimates are required, without the need for actually solving a particular problem. An example of such a situation is found in automated negotiation protocols regulating the exchange of personnel among different departments in a company or hospital. Such a situation may benefit from having quick performance estimates available. Instead of calculating and evaluating the individual objective functions, department managers can utilise predictive systems in order to speed up the negotiation time. Another application can be found in patient admission scheduling where the assignment of patients to rooms affects the coverage requirements. Predictive systems can facilitate the inclusion of such effects in the decision making process.

Using performance prediction as an algorithm component

From an engineering point of view, performance prediction models could also be employed by new algorithms. Such predictions can give an indication of the resulting quality on a given instance. This information can be used to point an algorithm towards a certain region in the search space. As a result, such an algorithm could use the allowed time to search deeper or more thoroughly in promising regions.

Investigating the relationship between features and algorithm performance

A deeper investigation of the relationship between the feature values and the performance measures of the algorithms could lead to a better insight into why certain algorithms perform well on certain instances. This is closely related to the field of algorithm footprints. In this dissertation, we have set a few first steps towards such an investigation. In the case study on nurse rostering, we discovered the crucial importance of the `ratioAvailabilityOverCoverage` feature. In the case study on project scheduling, we demonstrated the importance of known complexity indicators (characterising instances in terms of intrinsic hardness) for the empirical hardness of instances. While these are definitely important findings an sich, such knowledge could also be linked back to the algorithms. Investigating important features could expose certain regions of interest. Discovering the conditions under which an algorithm performs badly could lead to the development of better algorithms.

We must note here that this is touching an open question within the metaheuristics community. There is plenty of empirical evidence that metaheuristic methods work well on a range of hard problems encountered in both artificial intelligence and operational research. Most publications introduce a specific implementation of such a method in a new setting, or build an advanced technique improving some earlier results in an existing setting. There is however only limited knowledge on *why* these algorithms work well in certain circumstances, and why they do not in others. This topic is currently receiving more and more attention and is definitely a promising direction for further research.

Generalisation of the concepts of competitiveness and impact

In this dissertation, we introduced the concepts of competitiveness and potential impact in order to quantify the usefulness of building algorithm selection tools. We presented a possible definition/interpretation of these concepts for the case where only two algorithms are considered. Further research towards algorithm selection should generalise these concepts to allow for more algorithms. This could be done through pairwise comparisons, or by investigating individual measures indicating an algorithm's competitive position with respect to a group of other algorithms. Such generalisations are currently being investigated in our research team.

Appendix A

A feature set for nurse rostering problems

This appendix contains an extensive list of the NRP feature set developed for predicting algorithm performance for nurse rostering problems.

The 305 features in this set can be grouped into 5 categories: feature related to the problem size, features related to the coverage requirements, features related to the workforce structure, features related to the contract constraints and features related to the requests. All feature values can be calculated by simply iterating over the concepts in the problem description (i.e. in polynomial time). The calculation time for the instances considered in this study is a matter of a few milliseconds, which is negligible compared to the calculation time of the algorithms. The following list sums up all features. Whenever a concept can be weighted, the feature set includes both a weighted and an unweighted variant of the feature. The numbers between parentheses denote the number of unweighted features in the group.

- *features related to the problem size:* (6)
  - concerning the scheduling period:
    - the number of days in the planning period
    - the number of time units in the planning horizon
    - the number of different shift types
  - concerning the workforce:
    - the number of nurses

    - the number of distinct contract types
    - the number of skill types

- *features related to the coverage constraints:* (17)
  - the minimum, maximum, mean, standard deviation and variation of the number of required nurses
    - per day
    - per shift type
    - per skill type
  - the total number of shifts that need to be worked
  - the number of nurses divided by the total number of shifts that need to be worked

- *features related to the workforce structure:* (15)
  - the minimum, maximum, mean, standard deviation and variation of the number of
    - skill types per nurse
    - nurses per skill type
    - nurses per contract type

- *features related to the contract constraints:*
  - statistics on the constraint values: (45)
    the minimum, maximum, mean, standard deviation and variation over the contract types of:
    - the minimum number of assignments
    - the maximum number of assignments
    - the minimum number of consecutive working days
    - the maximum number of consecutive working days
    - the minimum number of consecutive free days
    - the maximum number of consecutive free days
    - the minimum number of consecutive working weekends
    - the maximum number of consecutive working weekends
    - the maximum number of working weekends in four weeks
  - ratios of constraint values: (12)
    the mean, minimum and maximum over the contract types of
    - the minimum-over-maximum ratio of
      - the number of assignments

        - the number of consecutive working days
        - the number of consecutive free days
        - the number of consecutive working weekends
    - special ratios concerning the tightness of the constraints: (9)
      the mean, minimum and maximum over the contract types of
        * the maximum number of assignments divided by the minimum number of consecutive working days
        * the minimum number of consecutive free days divided by the minimum number of consecutive working days
        * the number of shift types that can maximally be worked by the nurses (measured by their maximum number of assignments) divided by the total number of shifts that need to be worked
    - concerning boolean constraints: (4)
      the fraction of nurses that have the following boolean constraint set to true in their contract:
        * whether or not the working weekends should be complete
        * whether or not the nurse should work identical complete weekends
        * whether or not the nurse should have at least two days off after a series of night shifts
        * whether or nor the nurse should maximally work one shift a day
    - concerning the occurrence of patterns in a contract: (16)
        * the total number of patterns in the problem description
        * the minimum, maximum, mean, standard deviation and variation of the number of patterns per contract type
        * the minimum, maximum, mean, standard deviation and variation of the length of the patterns
        * the minimum, maximum, mean, standard deviation and variation of the weight of the patterns

- *features related to the requests:* (56)
    - the minimum, maximum, mean, standard deviation and variation over the nurses of
        * the total number of `On` requests (i.e. the sum of the number of `DayOn` and `ShiftOn` requests)
        * the total number of `Off` requests
        * the number of `DayOn` requests
        * the number of `DayOff` requests

    - the number of `ShiftOn` requests
    - the number of `ShiftOff` requests
- the minimum, maximum, mean, standard deviation and variation over the days of
    - the number of `On` requests
    - the number of `Off` requests
- the minimum, maximum, mean, standard deviation and variation over the shifts of
    - the number of `On` requests
    - the number of `Off` requests
- the total number of
    - `On` requests
    - `Off` requests
    - `DayOn` requests
    - `DayOff` requests
    - `ShiftOn` requests
    - `ShiftOff` requests

Appendix B

Reduced feature sets for nurse rostering

This appendix contains the lists of features used for building the prediction models throughout our case study on nurse rostering problems described in Section 3.3.2.

features related to the coverage constraints: (2)
`requiredNursesPerDay_variation`
`requiredNursesPerShift_maximum`
features related to the workforce structure: (1)
`skillsPerNurse_variation`
features related to the contract constraints:
statistics on the constraint values: (4)
`MinNumAssingments_stddev`
`MaxConsecutiveWorkingDays_variation`
`MinConsecutiveFreeDays:`
`mean, variation`
ratios of constraint values: (2)
`ratio_MaxMinNumAssignments:`
`mean, maximum`
special ratios concerning the tightness of the constraints: (1)
`ratio_MaxNumAssignments_MinConsecWorkingDays_maximum`
features related to the requests: (3)
`requestsOffPerShift:`
`mean, stddev, variation`

Table B.1: Reduced feature set for `Algorithm A` after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (19 features)

features related to the coverage constraints: (4)
`requiredNursesPerDay_minimum`
`requiredNursesPerShift:`
`stddev, variation, maximum`
features related to the contract constraints:
statistics on the constraint values: (7)
`MaxNumAssingments_maximum`
`MinNumAssingments_stddev`
`MaxConsecutiveWorkingDays_variation`
`MaxConsecutiveFreeDays_maximum`
`MinConsecutiveFreeDays:`
`mean, variation, minimum`
ratios of constraint values: (4)
`ratio_MaxMinNumAssignments_mean`
`ratio_MaxMinConsecutiveWorkingDays_maximum`
`ratio_MaxMinConsecutiveWorkingWeekends_mean`
`ratio_MaxMinConsecutiveFreeDays_mean`
special ratios concerning the tightness of the constraints: (1)
`ratio_MaxNumAssignments_MinConsecWorkingDays_maximum`
features related to the requests: (3)
`requestsOffPerShift:`
`mean, stddev, variation`

Table B.2: Reduced feature set for `Algorithm A` after applying backward correlation-based feature selection. (19 features)

features related to the coverage constraints: (2)
`requiredNursesPerDay_variation`
`requiredNursesPerShift_maximum`
features related to the workforce structure: (1)
`skillsPerNurse_variation`
features related to the contract constraints:
statistics on the constraint values: (3)
`MinNumAssingments_stddev`
`MinConsecutiveFreeDays:`
`mean, variation`
ratios of constraint values: (3)
`ratio_MaxMinNumAssignments:`
`mean, maximum`
`ratio_MaxMinConsecutiveWorkingWeekends_mean`
special ratios concerning the tightness of the constraints: (1)
`ratio_MaxNumAssignments_MinConsecWorkingDays_maximum`
features related to the requests: (3)
`requestsOffPerShift:`
`mean, stddev, variation`

Table B.3: Reduced feature set for `Algorithm B` after applying forward correlation-based feature selection. (13 features)

features related to the coverage constraints: (5)
`nrNursesPerShift`
`requiredNursesPerDay_minimum`
`requiredNursesPerShift:`
`stddev, variation, maximum`
features related to the contract constraints:
statistics on the constraint values: (8)
`MaxNumAssingments_maximum`
`MinNumAssingments_stddev`
`MaxConsecutiveWorkingDays_variation`
`MinConsecutiveWorkingWeekends_minimum`
`MaxConsecutiveFreeDays_maximum`
`MinConsecutiveFreeDays:`
`mean, variation, minimum`
ratios of constraint values: (4)
`ratio_MaxMinNumAssignments_maximum`
`ratio_MaxMinConsecutiveWorkingDays_maximum`
`ratio_MaxMinConsecutiveWorkingWeekends_mean`
`ratio_MaxMinConsecutiveFreeDays_mean`
special ratios concerning the tightness of the constraints: (1)
`ratio_MaxNumAssignments_MinConsecWorkingDays_maximum`
features related to the requests: (3)
`requestsOffPerShift:`
`mean, stddev, variation`

Table B.4: Reduced feature set for `Algorithm B` after applying backward correlation-based feature selection. (21 features)

features related to the coverage constraints: (2)
`requiredNursesPerDay_variation`
`requiredNursesPerShift_maximum`
features related to the workforce structure: (1)
`skillsPerNurse_variation`
features related to the contract constraints:
statistics on the constraint values: (4)
`MinNumAssingments_stddev`
`MaxConsecutiveWorkingDays_variation`
`MinConsecutiveFreeDays:`
`mean, variation`
ratios of constraint values: (3)
`ratio_MaxMinNumAssignments:`
`mean, maximum`
`ratio_MaxMinConsecutiveWorkingWeekends_mean`
special ratios concerning the tightness of the constraints: (1)
`ratio_MaxNumAssignments_MinConsecWorkingDays_maximum`
features related to the requests: (3)
`requestsOffPerShift:`
`mean, stddev, variation`

Table B.5: Reduced feature set for `Algorithm B` after applying backward correlation-based feature selection. (14 features)

features related to the problem size: (1)
`nrShifts`
features related to the coverage constraints: (4)
`requiredNursesPerShift:`
`stddev, maximum`
`requiredNursesPerSkill:`
`stddev, maximum`
features related to the workforce structure: (2)
`nrNursesPerContract:`
`stddev, maximum`
features related to the contract constraints:
statistics on the constraint values: (3)
`MaxNumAssingments_mean`
`MaxConsecutiveWorkingDays_mean`
`MinConsecutiveFreeDays_mean`
special ratios concerning the tightness of the constraints: (1)
`ratioAvailabilityOverCoverage`
features related to the requests: (4)
`requestsOffPerNurse_minimum`
`requestsShiftOffPerNurse_variation`
`requestsOffPerDay_maximum`
`requestsOffPerShift_variation`

Table B.6: Reduced feature set for `Algorithm A` after applying learner-dependent feature selection based on linear regression. (15 features)

features related to the problem size: (1)
`nrShifts`
features related to the coverage constraints: (4)
`requiredNursesPerShift:`
`stddev, maximum`
`requiredNursesPerSkill:`
`stddev, maximum`
features related to the workforce structure: (4)
`nrNursesPerSkill_maximum`
`nrNursesPerContract:`
`stddev, minimum, maximum`
features related to the contract constraints:
statistics on the constraint values: (10)
`MaxNumAssingments:`
`mean, maximum`
`MaxConsecutiveWorkingDays_mean`
`MinConsecutiveWorkingDays_maximum`
`MaxConsecutiveWorkingWeekends:`
`mean, minimum`
`MinConsecutiveWorkingWeekends_mean`
`MaxConsecutiveFreeDays_stddev`
`MinConsecutiveFreeDays:`
`mean, maximum`
special ratios concerning the tightness of the constraints: (1)
`ratioAvailabilityOverCoverage`
features related to the requests: (5)
`requestsOffPerNurse_minimum`
`requestsDayOffPerNurse_maximum`
`requestsShiftOffPerNurse_variation`
`requestsOffPerShift:`
`maximum, variation`

Table B.7: Reduced feature set for `Algorithm B` after applying learner-dependent feature selection based on linear regression. (25 features)

Appendix C

A feature set for project scheduling problems

This appendix contains an extensive list of the feature set developed for the construction of empirical hardness models for the (multi-mode) resource-constrained project scheduling problem.

Resource types can be renewable, non-renewable or doubly-constrained. The latter type is however not used in the benchmark sets in this dissertation because it can be represented by the combination of a renewable and a non-renewable resource type. Therefore, we did not always include doubly-constrained resource types in the features. Nevertheless, the feature set can easily be extended to include similar features regarding doubly-constrained resource types.

For the basic version of the problem (single-mode), the features that take the mean, minimum and maximum over the modes will all have the same values, features that take the standard deviation over the modes will have value 0 and features that take the variation will either be 0 of not defined (NaN). Similarly, when no non-renewable resources are used, a large set of features will also become redundant.

Currently, the feature set consists of 686 instance features for the MRCPSP. These features can be grouped into four categories: size related, resource constraint related, precedence constraint related and activity duration related features.

Note that a subset of (generalisations of) the complexity measures of Section 4.2.1 is also included in this set, albeit under a different name.

The following list sums up all features: (the number of features in each group is also given)

- *features related to the problem size:* (35)
  - the number of activities
  - the total number of resource types, which is the sum of three other features:
    - the number of renewable resource types
    - the number of non-renewable resource types
    - the number of doubly constrained resource types
  - the total number of resource units available, which is the sum of
    - the number of renewable resource units available
    - the number of non-renewable resource units available
    - the number of doubly constrained resource units available
  - the ratio of the total number of resource types over the number of activities
  - the ratio of the total number of resource units available over the number of activities
  - concerning the composition of the resource pool:
    - the fraction of renewable resource types (the number of renewable resource types over the total number of resource types)
    - the fraction of renewable resource units available (the number of renewable resource units available over the total number of resource types available)
    - the fraction of non-renewable resource types
    - the fraction of non-renewable resource units available
    - the fraction of doubly constrained resource types
    - the fraction of doubly constrained resource units available
  - for each resource type, the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of the number of resource units available
    - Additionally, this is done for the renewable and non-renewable resource types separately.

- *features related to the resource constraints:* (486)
  - Features which are aggregated over the activities:
    First of all, we look at the number of resource types, as well as the number of resource units each activity requires. As each activity possibly has several modes, we calculate for each activity the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum over the modes of:
    - the number of renewable resource types required
    - the number of renewable resource units required
    - the number of non-renewable resource types required
    - the number of non-renewable resource units required
    - the total number of resource types required
    - the total number of resource units required

    Afterwards, we aggregate this information over the activities by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, resulting in 216 features (6*6*6 = 216). These features include (various statistics on) the resource factor of each resource type as defined by Pascoe (1966).
  - Features which are aggregated over the resource types:
    Again, we look at the number of resource units each activity requires. Each activity has possibly several modes, we consider these modes separately and calculate for each resource type the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of required resource units per mode.
    Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).
    We also look at the number of activities needing resources of a certain type. Since activities have several modes, we first calculate for each resource type and each activity the fraction of modes of this activity that requires this resource type. Then the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum over all activities of this fraction is calculated.
    Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these

values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).

- Ratios of feature values (related to the resource constrainedness as defined by Patterson (1976)).

  We look at the number of resource units each activity requires. We use the mean, minimum and maximum number of units over the modes of each activity (as already calculated above). We add up these values for all activities and divide this by the available resource units of each type.

  Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This leads to another 54 features (3*6*3). Note that this information is related to the (inverse of) the resource strength as originally defined by Cooper (1976).

- *features related to the precedence constraints:* (21)

  (These features are independent of the existence of modes, hence we include the same features as for the RCPSP)

  - the total number of precedence constraints
  - the number of precendence constraints divided by the theoretical maximum number of precedence constraints (defined as order strength by Mastor (1970))
  - the ratio of the number of precedence constraints over the number of activities (which is the coefficient of network complexity as defined by Pascoe (1966))
  - the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of
    - the total number of precedence constraints per activity
    - the number of predecessors per activity
    - the number of successors per activity

- *features related to the activity durations:* (144)

  - Features which are aggregated over the activities:

    Each activity can have several modes, so we first calculate the mean, standard deviation, variation, minimum, maximum and the ratio of

the minimum over the maximum of the duration of the modes per activity. Afterwards, we aggregate this information over the activities. We calculate the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values over the activities, resulting in 36 features.

– Features which are aggregated over the resource types:

  Again, we look at the duration of the activities that require certain resources. Each activity has possibly several modes, we consider these modes separately and calculate for each resource type the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of the duration of modes needing this resource.

  Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).

Note that we do not include the complexity index defined by De Reyck and Herroelen (1996). The computation of this measure is not as straightforward and efficient as the other features in this set. The resource strength defined by Kolisch et al. (1995) is also left out for the reasons explained in Section 4.2.1.

Appendix D

Reduced feature sets for project scheduling

This appendix contains the lists of features used for building the prediction models throughout our case study on project scheduling problems described in Section 4.3.

features related to the resource constraints: (14)
`renewableResourceUnitsPerMode_variationPerAct_stddev`
`renewableResourceUnitsPerMode_minOverMaxPerAct:`
`mean, minOverMax`
`resourceUnitsPerMode_stddevPerAct_stddev`
`resourceUnitsPerMode_minOverMaxPerAct_minOverMax`
`ratioMinRequestedOverAvailability_mean`
`ratioMinNonRenewableOverAvailability:`
`mean, stddev`
`ratioMaxRequestedOverAvailability_minimum`
`ratioMaxRenewableOverAvailability_stddev`
`ratioMaxNonRenewableOverAvailability:`
`variation, maximum`
`ratioMeanRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (4)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct_variation`
`predecessorsPerAct_variation`
`successorsPerAct_variation`
features related to the activity durations: (10)
`durationPerMode_stddev_variation`
`durationPerMode_minimum:`
`variation, minOverMax`
`durationPerMode_maximum_variation`
`durationPerMode_minOverMax:`
`mean, variation, minimum, minOverMax`
`durationPerRenewableType_variation_maximum`
`durationPerType_variation_maximum`

Table D.1: Reduced feature set for `Algorithm A` (5000 schedules) after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (28 features)

features related to the resource constraints: (33)
`renewableResourceUnitsPerMode_stddevPerAct_mean`
`nonRenewableResourceUnitsPerMode_stddevPerAct_stddev`
`resourceUnitsPerMode_stddevPerAct:`
`stddev, variation`
`resourceUnitsPerMode_variationPerAct_variation`
`resourceUnitsPerMode_minOverMaxPerAct_minOverMax`
`resourceUnitsPerNonRenewableType_stddev_mean`
`resourceUnitsPerNonRenewableType_maximum_mean`
`resourceUnitsPerType_stddev_mean`
`ratioMinRequestedOverAvailability:`
`mean, minimum, maximum`
`ratioMinRenewableOverAvailability_minimum`
`ratioMinNonRenewableOverAvailability:`
`mean, stddev, variation, minOverMax`
`ratioMaxRequestedOverAvailability:`
`mean, stddev, minimum`
`ratioMaxRenewableOverAvailability:`
`stddev, minimum`
`ratioMaxNonRenewableOverAvailability:`
`mean, stddev, variation, minimum, maximum, minOverMax`
`ratioMeanRequestedOverAvailability:`
`mean, stddev, minimum`
`ratioMeanRenewableOverAvailability_stddev`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (5)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct_variation`
`predecessorsPerAct_variation`
`successorsPerAct:`
`variation, maximum`
features related to the activity durations: (7)
`durationPerMode_mean_maximum`
`durationPerMode_stddev_mean`
`durationPerMode_minimum_minOverMax`
`durationPerMode_maximum_variation`
`durationPerMode_minOverMax:`
`variation, minimum, minOverMax`

Table D.2: Reduced feature set for `Algorithm A` (5000 schedules) after applying backward correlation-based feature selection. (45 features)

features related to the resource constraints: (15)
`renewableResourceUnitsPerMode_variationPerAct_mean`
`nonRenewableResourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerMode_stddevPerAct_stddev`
`resourceUnitsPerMode_minOverMaxPerAct_minOverMax`
`ratioMinRequestedOverAvailability:`
`mean, minimum`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMaxRenewableOverAvailability_stddev`
`ratioMaxNonRenewableOverAvailability:`
`mean, stddev, variation, maximum, minOverMax`
`ratioMeanRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (5)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct:`
`variation, minOverMax`
`predecessorsPerAct_variation`
`successorsPerAct_variation`
features related to the activity durations: (9)
`durationPerMode_stddev:`
`stddev, variation`
`durationPerMode_variation_mean`
`durationPerMode_minimum_minOverMax`
`durationPerMode_minOverMax:`
`mean, variation, minimum, minOverMax`
`durationPerRenewableType_variation_maximum`

Table D.3: Reduced feature set for `Algorithm B` (5000 schedules) after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (29 features)

features related to the resource constraints: (30)
renewableResourceUnitsPerMode_stddevPerAct_mean
nonRenewableResourceUnitsPerMode_stddevPerAct_stddev
nonRenewableResourceUnitsPerMode_minimumPerAct_stddev
resourceUnitsPerMode_stddevPerAct:
stddev, variation
resourceUnitsPerMode_variationPerAct_variation
resourceUnitsPerMode_minOverMaxPerAct_minOverMax
resourceUnitsPerNonRenewableType_maximum_mean
ratioMinRequestedOverAvailability:
mean, minimum, maximum
ratioMinRenewableOverAvailability_minimum
ratioMinNonRenewableOverAvailability:
mean, stddev, variation, minOverMax
ratioMaxRequestedOverAvailability:
mean, stddev, minimum
ratioMaxRenewableOverAvailability:
stddev, minimum
ratioMaxNonRenewableOverAvailability:
mean, stddev, variation, minimum, maximum, minOverMax
ratioMeanRequestedOverAvailability:
mean, minimum
ratioMeanNonRenewableOverAvailability_mean
features related to precedence constraints: (5)
nrPrecedenceConstraintsOverTheoreticalMaximum
precedenceConstraintsPerAct_variation
predecessorsPerAct_variation
successorsPerAct:
variation, maximum
features related to the activity durations: (8)
durationPerMode_mean:
stddev, maximum
durationPerMode_stddev_maximum
durationPerMode_minimum_minOverMax
durationPerMode_maximum_variation
durationPerMode_minOverMax:
variation, minimum, minOverMax

Table D.4: Reduced feature set for `Algorithm B` (5000 schedules) after applying backward correlation-based feature selection. (43 features)

features related to the resource constraints: (12)
`renewableResourceUnitsPerMode_stddevPerAct_variation`
`renewableResourceUnitsPerMode_minOverMaxPerAct:`
`mean, minOverMax`
`resourceUnitsPerMode_stddevPerAct_stddev`
`resourceUnitsPerMode_minOverMaxPerAct_minOverMax`
`ratioMinRequestedOverAvailability_mean`
`ratioMinNonRenewableOverAvailability:`
`mean, stddev`
`ratioMaxNonRenewableOverAvailability:`
`mean, maximum`
`ratioMeanRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (4)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct_variation`
`predecessorsPerAct_variation`
`successorsPerAct_variation`
features related to the activity durations: (8)
`durationPerMode_minimum:`
`variation, minOverMax`
`durationPerMode_maximum_variation`
`durationPerMode_minOverMax:`
`mean, variation, minimum, minOverMax`
`durationPerRenewableType_variation_maximum`

Table D.5: Reduced feature set for `Algorithm A` (25000 schedules) after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (24 features)

features related to the resource constraints: (32)
`renewableResourceUnitsPerMode_stddevPerAct:`
`mean, variation`
`nonRenewableResourceUnitsPerMode_stddevPerAct_stddev`
`resourceUnitsPerMode_stddevPerAct:`
`stddev, variation`
`resourceUnitsPerMode_variationPerAct_variation`
`resourceUnitsPerMode_minOverMaxPerAct_minOverMax`
`resourceUnitsPerNonRenewableType_stddev_mean`
`resourceUnitsPerNonRenewableType_maximum_mean`
`resourceUnitsPerType_stddev_mean`
`ratioMinRequestedOverAvailability:`
`mean, minimum, maximum`
`ratioMinRenewableOverAvailability_minimum`
`ratioMinNonRenewableOverAvailability:`
`mean, stddev, variation, minOverMax`
`ratioMaxRequestedOverAvailability:`
`mean, stddev, minimum`
`ratioMaxRenewableOverAvailability:`
`stddev, minimum`
`ratioMaxNonRenewableOverAvailability:`
`mean, stddev, variation, minimum, maximum, minOverMax`
`ratioMeanRequestedOverAvailability:`
`mean, minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (5)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct_variation`
`predecessorsPerAct_variation`
`successorsPerAct:`
`variation, maximum`
features related to the activity durations: (5)
`durationPerMode_mean_maximum`
`durationPerMode_minimum_minOverMax`
`durationPerMode_minOverMax:`
`variation, minimum, minOverMax`

Table D.6: Reduced feature set for `Algorithm A` (25000 schedules) after applying backward correlation-based feature selection. (42 features)

features related to the resource constraints: (21)
`renewableResourceUnitsPerMode_variationPerAct_minimum`
`renewableResourceUnitsPerMode_minOverMaxPerAct:`
`mean, minOverMax`
`resourceUnitsPerMode_stddevPerAct_variation`
`resourceUnitsPerMode_variationPerAct:`
`variation, minimum`
`resourceUnitsPerRenewableType_maximum_variation`
`resourceUnitsPerType_stddev_maximum`
`ratioMinRequestedOverAvailability:`
`mean, minimum`
`ratioMinNonRenewableOverAvailability:`
`mean, stddev`
`ratioMaxRequestedOverAvailability_minimum`
`ratioMaxRenewableOverAvailability_stddev`
`ratioMaxNonRenewableOverAvailability:`
`mean, stddev, variation, maximum, minOverMax`
`ratioMeanRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (5)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`precedenceConstraintsPerAct:`
`variation, minOverMax`
`predecessorsPerAct_variation`
`successorsPerAct_variation`

Table D.7: Reduced feature set for `Algorithm B` (25000 schedules) after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (26 features)

features related to the resource constraints: (32)
renewableResourceUnitsPerMode_stddevPerAct:
stddev, variation
nonRenewableResourceUnitsPerMode_stddevPerAct_stddev
resourceUnitsPerMode_stddevPerAct:
stddev, variation
resourceUnitsPerMode_variationPerAct_variation
resourceUnitsPerMode_minOverMaxPerAct:
stddev, minOverMax
resourceUnitsPerNonRenewableType_maximum_mean
ratioMinRequestedOverAvailability:
mean, minimum, maximum
ratioMinRenewableOverAvailability_minimum
ratioMinNonRenewableOverAvailability:
mean, stddev, variation, minOverMax
ratioMaxRequestedOverAvailability:
mean, stddev, minimum
ratioMaxRenewableOverAvailability:
stddev, minimum
ratioMaxNonRenewableOverAvailability:
mean, stddev, variation, minimum, maximum, minOverMax
ratioMeanRequestedOverAvailability:
mean, minimum
ratioMeanRenewableOverAvailability_stddev
ratioMeanNonRenewableOverAvailability_mean
features related to precedence constraints: (5)
nrPrecedenceConstraintsOverTheoreticalMaximum
precedenceConstraintsPerAct_variation
predecessorsPerAct_variation
successorsPerAct:
variation, maximum
features related to the activity durations: (6)
durationPerMode_mean_maximum
durationPerMode_stddev:
mean, maximum
durationPerMode_variation_maximum
durationPerMode_minimum_minOverMax
durationPerMode_minOverMax_minOverMax

Table D.8: Reduced feature set for `Algorithm B` (25000 schedules) after applying backward correlation-based feature selection. (43 features)

features related to the problem size: (2)
`totalNrResourceUnitsAvailableOverNrActivities`
`resourceUnitsAvailablePerType_minOverMax`
features related to the resource constraints: (19)
`nonRenewableResourceTypesPerMode_meanPerAct_mean`
`nonRenewableResourceUnitsPerMode_meanPerAct_maximum`
`nonRenewableResourceUnitsPerMode_stddevPerAct_variation`
`nonRenewableResourceUnitsPerMode_variationPerAct_stddev`
`nonRenewableResourceUnitsPerMode_maximumPerAct_stddev`
`nonRenewableResourceUnitsPerMode_minOverMaxPerAct_maximum`
`resourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerNonRenewableType_mean_mean`
`resourceUnitsPerNonRenewableType_stddev_stddev`
`resourceUnitsPerNonRenewableType_minimum_mean`
`resourceUnitsPerType_stddev_maximum`
`resourceUnitsPerType_minOverMax_mean`
`ratioMinRequestedOverAvailability:` `mean, stddev`
`ratioMinRenewableOverAvailability_minimum`
`ratioMaxRenewableOverAvailability:` `stddev, minimum`
`ratioMeanRequestedOverAvailability_minOverMax`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (2)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`predecessorsPerAct_variation`
features related to the activity durations: (2)
`durationPerMode_mean_variation`
`durationPerRenewableType_variation_minimum`

Table D.9: Reduced feature set for `Algorithm A` (5000 schedules) after applying forward linear regression-based feature selection. (25 features)

features related to the problem size: (5)
`totalNrResourceTypes`
`totalNrResourceUnitsAvailableOverNrActivities`
`resourceUnitsAvailablePerRenewableType:` `stddev, variation`
`resourceUnitsAvailablePerType_minOverMax`
features related to the resource constraints: (16)
`renewableResourceTypesPerMode_meanPerAct_minimum`
`renewableResourceUnitsPerMode_meanPerAct_minOverMax`
`renewableResourceUnitsPerMode_maximumPerAct_minOverMax`
`nonRenewableResourceTypesPerMode_meanPerAct:` `mean, maximum`
`nonRenewableResourceUnitsPerMode_variationPerAct_minimum`
`resourceUnitsPerMode_stddevPerAct_mean`
`resourceUnitsPerMode_minimumPerAct_minOverMax`
`resourceUnitsPerMode_minOverMaxPerAct_maximum`
`resourceUnitsPerNonRenewableType_stddev:` `mean, stddev`
`resourceUnitsPerType_stddev_maximum`
`resourceUnitsPerType_minOverMax_mean`
`ratioMinRequestedOverAvailability_mean`
`ratioMaxRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (2)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`predecessorsPerAct_variation`
features related to the activity durations: (4)
`durationPerMode_mean_maximum`
`durationPerMode_minimum_variation`
`durationPerRenewableType_variation_minimum`
`durationPerNonRenewableType_stddev_stddev`

Table D.10: Reduced feature set for `Algorithm B` (5000 schedules) after applying forward linear regression-based feature selection. (27 features)

features related to the problem size: (3)
`totalNrResourceTypes`
`totalNrResourceUnitsAvailableOverNrActivities`
`resourceUnitsAvailablePerType_variation`
features related to the resource constraints: (19)
`nonRenewableResourceUnitsPerMode_stddevPerAct_mean`
`resourceTypesPerMode_stddevPerAct_mean`
`resourceUnitsPerMode_minimumPerAct_minOverMax`
`resourceUnitsPerNonRenewableType_stddev_stddev`
`resourceUnitsPerNonRenewableType_maximum_mean`
`resourceUnitsPerType_stddev_maximum`
`resourceUnitsPerType_variation_mean`
`resourceUnitsPerType_maximum_stddev`
`resourceUnitsPerType_minOverMax_mean`
`ratioMinRequestedOverAvailability:` `mean, minimum`
`ratioMinRenewableOverAvailability:` `variation, minimum`
`ratioMinNonRenewableOverAvailability_stddev`
`ratioMaxRenewableOverAvailability_minimum`
`ratioMaxNonRenewableOverAvailability:` `minimum, maximum`
`ratioMeanRequestedOverAvailability_mean`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (2)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`predecessorsPerAct_variation`
features related to the activity durations: (4)
`durationPerMode_minimum_variation`
`durationPerNonRenewableType_mean_stddev`
`durationPerNonRenewableType_variation_minimum`
`durationPerType_variation_minimum`

Table D.11: Reduced feature set for `Algorithm A` (25000 schedules) after applying forward linear regression-based feature selection. (28 features)

features related to the problem size: (2)
`nrActivities`
`resourceUnitsAvailablePerRenewableType_variation`
features related to the resource constraints: (8)
`renewableResourceUnitsPerMode_variationPerAct_maximum`
`nonRenewableResourceUnitsPerMode_stddevPerAct_variation`
`resourceUnitsPerNonRenewableType_stddev_stddev`
`resourceUnitsPerType_stddev_maximum`
`resourceUnitsPerType_minOverMax_mean`
`ratioMinRequestedOverAvailability_mean`
`ratioMaxRequestedOverAvailability_minOverMax`
`ratioMeanNonRenewableOverAvailability_mean`
features related to precedence constraints: (2)
`nrPrecedenceConstraintsOverTheoreticalMaximum`
`predecessorsPerAct_variation`
features related to the activity durations: (2)
`durationPerRenewableType_variation_mean`
`durationPerNonRenewableType_stddev_stddev`

Table D.12: Reduced feature set for `Algorithm B` (25000 schedules) after applying forward linear regression-based feature selection. (14 features)

features related to the problem size: (1)
`resourceUnitsAvailablePerNonRenewableType_mean`
features related to the resource constraints: (19)
`renewableResourceUnitsPerMode_minOverMaxPerAct:`
`mean, minimum`
`nonRenewableResourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerMode_minOverMaxPerAct:`
`mean, minOverMax`
`resourceUnitsPerType_maximum_mean`
`resourceUnitsPerType_minOverMax:`
`mean, maximum`
`ratioMinRequestedOverAvailability:`
`variation, minOverMax`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMaxRequestedOverAvailability:`
`mean, stddev, variation`
`ratioMaxRenewableOverAvailability_minimum`
`ratioMaxNonRenewableOverAvailability_mean`
`ratioMeanRequestedOverAvailability:`
`mean, minOverMax`
`ratioMeanNonRenewableOverAvailability_mean`
features related to the activity durations: (2)
`durationPerMode_mean_minimum`
`durationPerMode_variation_stddev`

Table D.13: Reduced feature set for `AS2` (5000 schedules) after applying forward correlation-based feature selection. (22 features)

features related to the problem size: (1)
`resourceUnitsAvailablePerNonRenewableType_mean`
features related to the resource constraints: (24)
`renewableResourceUnitsPerMode_stddevPerAct_stddev`
`renewableResourceUnitsPerMode_minimumPerAct_variation`
`renewableResourceUnitsPerMode_minOverMaxPerAct:`
`mean, minimum`
`nonRenewableResourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerMode_minOverMaxPerAct:`
`mean, minOverMax`
`resourceUnitsPerNonRenewableType_minOverMaxPerAct_maximum`
`resourceUnitsPerType_maximum_mean`
`resourceUnitsPerType_minOverMax:`
`mean, maximum`
`ratioMinRequestedOverAvailability:`
`variation, minimum, minOverMax`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMaxRequestedOverAvailability:`
`mean, stddev, variation, minOverMax`
`ratioMaxRenewableOverAvailability_minimum`
`ratioMaxNonRenewableOverAvailability_mean`
`ratioMeanRequestedOverAvailability:`
`minimum, minOverMax`
`ratioMeanNonRenewableOverAvailability_mean`
features related to the activity durations: (3)
`durationPerMode_mean_minimum`
`durationPerMode_variation:`
`stddev, minOverMax`

Table D.14: Reduced feature set for `AS2` (5000 schedules) after applying backward correlation-based feature selection. (28 features)

features related to the problem size: (1)
`resourceUnitsAvailablePerNonRenewableType_mean`
features related to the resource constraints: (15)
`renewableResourceUnitsPerMode_minOverMaxPerAct:` `mean, minimum`
`nonRenewableResourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerMode_minOverMaxPerAct_mean`
`resourceUnitsPerType_maximum_mean`
`resourceUnitsPerType_minOverMax:` `mean, maximum`
`ratioMinRequestedOverAvailability_minOverMax`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMaxRequestedOverAvailability:` `mean, variation`
`ratioMaxRenewableOverAvailability_minimum`
`ratioMeanRequestedOverAvailability:` `minimum, minOverMax`
`ratioMeanNonRenewableOverAvailability_mean`
features related to the activity durations: (1)
`durationPerMode_variation_stddev`

Table D.15: Reduced feature set for `AS2` (5000 schedules) after applying bi-directional correlation-based feature selection. (17 features)

features related to the problem size: (2)
`totalNrResourceUnitsAvailable`
`resourceUnitsAvailablePerNonRenewableType_mean`
features related to the resource constraints: (10)
`fractionOfModesNeedingNonRenewable_stddev_stddev`
`resourceUnitsPerType_variation_mean`
`resourceUnitsPerType_minOverMax_maximum`
`ratioMinRequestedOverAvailability:` `minimum, maximum, minOverMax`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMeanRequestedOverAvailability_minimum`
`ratioMeanNonRenewableOverAvailability:` `mean, minOverMax`
features related to precedence constraints: (1)
`nrPrecedenceConstraints`

Table D.16: Reduced feature set for `AS2` (25000 schedules) after applying forward correlation-based feature selection. (Identical for bi-directional correlation-based feature selection.) (13 features)

features related to the problem size: (2)
`totalNrResourceUnitsAvailable`
`resourceUnitsAvailablePerNonRenewableType_mean`
features related to the resource constraints: (11)
`fractionOfModesNeedingNonRenewable_stddev_stddev`
`resourceUnitsPerType_variation_mean`
`resourceUnitsPerType_minOverMax_maximum`
`ratioMinRequestedOverAvailability:`
`minimum, maximum`
`ratioMinNonRenewableOverAvailability_mean`
`ratioMeanRequestedOverAvailability:`
`minimum, minOverMax`
`ratioMeanNonRenewableOverAvailability:`
`mean, variation, minOverMax`
features related to precedence constraints: (1)
`nrPrecedenceConstraintsOverNrActivities`

Table D.17: Reduced feature set for `AS2` (25000 schedules) after applying backward correlation-based feature selection. (14 features)

Bibliography

Adenso-Diaz B. and Laguna M. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

Almajano P., Cerquides J., and Rodriguez-Aguilar J. A. Empirical hardness for mixed auctions. In *Current Topics in Artificial Intelligence*, pages 161–170. Springer, 2010.

Alvarez-Valdes R. and Tamarit J. M. Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis. In Slowinski R. and Węglarz J., editors, *Advances in Project Scheduling*, pages 113–134. Elsevier, 1989.

Angel E. and Zissimopoulos V. On the classification of np-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99(1): 261–277, 2000.

Ansótegui C., Sellmann M., and Tierney K. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, pages 142–157, 2009.

Bachelet V. *Métaheuristiques parallèles hybrides: application au problème d'affectation quadratique*. PhD thesis, Université de Lille 1, 1999.

Balas E. and Zemel E. An algorithm for large zero-one knapsack problems. *operations Research*, 28(5):1130–1154, 1980.

Bard J. F. and Purnomo H. W. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164(2):510–534, 2005.

Battiti R. and Protasi M. Reactive local search for the maximum clique problem 1. *Algorithmica*, 29(4):610–637, 2001.

Bein W. W., Kamburowski J., and Stallmann M. F. M. Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 21: 1112–1129, 1992.

Berrada I., Ferland J. A., and Michelon P. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3):183–193, 1996.

Bierwirth C., Mattfeld D. C., and Watson J.-P. Landscape regularity and random walks for the job-shop scheduling problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 21–30. Springer, 2004.

Bilgin B. *Advanced models and solution methods for automation of personnel rostering optimisation.* PhD thesis, KU Leuven, 2012.

Bilgin B., Demeester P., Misir M., Vancroonenburg W., Vanden Berghe G., and Wauters T. A hyper-heuristic with a greede shuffle approach to the nurse rostering competition. https://www.kuleuven-kulak.be/~u0041139/nrpcompetition/abstracts/s5.pdf, 2010.

Bilgin B., Demeester P., Misir M., Vancroonenburg W., and Vanden Berghe G. One hyperheuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434, 2012.

Birattari M. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective.* PhD thesis, Université Libre de Bruxelles, 2004.

Birattari M., Stützle T., Paquete L., and Varrentrapp K. A racing algorithm for configuring metaheuristics. In *Proceedings of the genetic and evolutionary computation conference*, pages 11–18. Citeseer, 2002.

Blazewicz J., Lenstra J. K., and Rinnooy Kan A. H. G. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

Borenstein Y. and Poli R. Kolmogorov complexity, optimization and hardness. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 112–119. IEEE, 2006.

Brucker P., Drexl A., Möhring R., Neumann K., and Pesch E. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

Buddhakulsomsiri J. and Kim D. S. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research*, 178(2):374–390, 2007.

Burke E., De Causmaecker P., and Vanden Berghe G. A hybrid tabu search algorithm for the nurse rostering problem. In McKay B., Yao X., Newton C., Kim J.-H., and Furuhashi T., editors, *Simulated Evolution and Learning*, volume 1585 of *Lecture Notes in Computer Science*, pages 187–194. Springer Berlin Heidelberg, 1999.

Burke E., Kendall G., Newall J., Hart E., Ross P., and Schulenburg S. Hyper-heuristics: An emerging direction in modern search technology. In Glover F. and Kochenberger G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 457–474. Springer, 2003.

Burke E., Curtois T., Post G., Qu R., and Veltman B. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188:330–341, 2008.

Burke E. K., De Causmaecker P., Petrovic S., and Vanden Berghe G. Fitness evaluation for nurse scheduling problems. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 2, pages 1139–1146, 2001.

Burke E. K., De Causmaecker P., and Vanden Berghe G. Novel meta-heuristic approaches to nurse rostering problems in belgian hospitals. *Handbook of scheduling: algorithms, models and performance analysis*, pages 44–1, 2004.

Burke E. K., Hyde M., Kendall G., Ochoa G., Özcan E., and Woodward J. R. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010.

Chang C. and Lee R. C. *Symbolic logic and mechanical theorem proving*, volume 67. Academic press New York, 1973.

Cheeseman P., Kanefsky B., and Taylor W. M. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, IJCAI, pages 331–337, 1991.

Cho Y. K., Moore J. T., Hill R. R., and Reilly C. H. Exploiting empirical knowledge for bi-dimensional knapsack problem heuristics. *International Journal of Industrial and Systems Engineering*, 3(5):530–548, 2008.

Chung F. R. *Spectral Graph Theory*, volume 92. American Mathematical Society, 1997.

Cooper D. F. Heuristics for scheduling resource-constrained projects: An experimental investigation. *Management Science*, 22(11):1186–1194, 1976.

Corne D. W. and Reynolds A. P. Optimisation and generalisation: Footprints in instance space. In Schaefer R., Cotta C., Kolodziej J., and Rudolph G., editors, *PPSN (1)*, volume 6238 of *Lecture Notes in Computer Science*, pages 22–31. Springer, 2010.

Dar-El E. M. MALB - a heuristic technique for balancing large single-model assembly lines. *AIIE Transactions*, 5(4):343–356, 1973.

Davis E. W. Project network summary measures constrained resource scheduling. *AIIE Transactions*, 7(2):132–142, 1975.

Davis M., Logemann G., and Loveland D. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

De Causmaecker P. and Vanden Berghe G. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

De Reyck B. and Herroelen W. On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 91(2):347–366, 1996.

Demeulemeester E. L. and Herroelen W. S. *Project scheduling: a research handbook*. Kluwer Academic Publishers, 2002.

Denzinger J., Fuchs M., and Fuchs M. High performance atp systems by combining several ai methods. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'97, pages 102–107, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

Eiben Á. E., Van Der Hauw J. K., and van Hemert J. I. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

Elmaghraby S. E. and Herroelen W. S. On the measurement of complexity in activity networks. *European Journal of Operational Research*, 5(4):223–234, 1980.

Ernst A. T., Jiang H., Krishnamoorthy M., and Sier D. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

Gagliolo M. and Schmidhuber J. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006.

Gent I. P. and Walsh T. Csplib: a benchmark library for constraints. In *Principles and Practice of Constraint Programming–CP'99*, pages 480–481. Springer, 1999.

Glover F. and McMillan C. The general employee scheduling problem. an integration of ms and ai. *Computers & operations research*, 13(5):563–573, 1986.

Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., and Witten I. H. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11: 10–18, 2009.

Hall N. G. and Posner M. E. Performance prediction and preselection for optimization and heuristic solution procedures. *Operations research*, 55(4): 703–716, 2007.

Hartmann S. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL)*, 49(5):433–448, 2002.

Hartmann S. and Briskorn D. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

Hartmann S. and Kolisch R. Experimental evaluation of state-of-the-art heuristics for the resourc-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, 1998.

Haspeslagh S. *Negotiation at the short and mid term level supported by analysis of nurse rostering problems.* PhD thesis, KU Leuven, 2012.

Haspeslagh S. and De Causmaecker P. Bridging the gap between short and mid term nurse rostering through a negotiation protocol. submitted, 2013.

Haspeslagh S., De Causmaecker P., and Vanden Berghe G. Distributed decision making in hospital wide nurse rostering problems. In Baptiste P., Kendall G., Munier-Kordon A., and Sourd F., editors, *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007), August 28-31 2007, Paris, France*, pages 192–199, 2007.

Haspeslagh S., De Causmaecker P., Schaerf A., and Stølevik M. The first international nurse rostering competition 2010. *Annals of Operations Research*, pages 1–16, 2012.

Haspeslagh S., Messelis T., Vanden Berghe G., and De Causmaecker P. An efficient translation scheme for representing nurse rostering problems as satisfiability problems. to appear in Proceedings of the 5th International Conference on Agents and Artificial Intelligence, February 2013, Barcelona, Spain, 2013.

Herroelen W. and De Reyck B. Phase transitions in project scheduling. *Journal of the Operational Research Society*, 50(2):148–156, 1999.

Herroelen W., Demeulemeester E., and De Reyck B. A classification scheme for project scheduling. In Weglarz J., editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 1–26. Kluwer, 1998.

Hill R. R. and Reilly C. H. The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. *Management Science*, 46(2):302–317, 2000.

Hsu E. I., Muise C. J., Beck J. C., and McIlraith S. A. Probabilistically estimating backbones and variable bias: Experimental overview. In *Principles and Practice of Constraint Programming*, pages 613–617. Springer, 2008.

Hutter F., Tompkins D. A. D., and Hoos H. H. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2002.

Hutter F., Hoos H. H., and Stützle T. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence-Volume 2*, pages 1152–1157. AAAI Press, 2007.

Hutter F., Hoos H. H., Leyton-Brown K., and Murphy K. P. An experimental investigation of model-based parameter optimisation: Spo and beyond. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 271–278. ACM, 2009a.

Hutter F., Hoos H. H., Leyton-Brown K., and Stützle T. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009b.

Hutter F., Hoos H. H., and Leyton-Brown K. Automated configuration of mixed integer programming solvers. In Lodi A., Milano M., and Toth P., editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer Berlin Heidelberg, 2010.

Hutter F., Hoos H. H., and Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Hutter F., Xu L., Hoos H. H., and Leyton-Brown K. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206(0):79 – 111, 2014.

Ivančić F., Yang Z., Ganai M. K., Gupta A., Shlyakhter I., and Ashar P. F-soft: Software verification platform. In *Computer Aided Verification*, pages 301–306. Springer, 2005.

Jones T. and Forrest S. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th international conference on genetic algorithms*, pages 184–192. Citeseer, 1995.

Kadioglu S., Malitsky Y., Sellmann M., and Tierney K. ISAC - instance-specific algorithm configuration. In *ECAI'10*, pages 751–756, 2010.

Kao E. P. C. and Queyranne M. On dynamic programming methods for assembly line balancing. *Operations Research*, 30(2):375–390, 1982.

Kohonen T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

Kolisch R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320 – 333, 1996.

Kolisch R. and Drexl A. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics (NRL)*, 43(1):23–40, 1996.

Kolisch R. and Drexl A. Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, 29:987–999, 1997.

Kolisch R. and Sprecher A. PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

Kolisch R., Sprecher A., and Drexl A. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703, 1995.

Kostuch P. and Socha K. Hardness prediction for the university course timetabling problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 135–144. Springer, 2004.

Kotthoff L. *On algorithm selection, with an application to combinatorial search problems*. PhD thesis, University of St Andrews, 2012.

Leyton-Brown K., Nudelman E., and Shoham Y. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In Hentenryck P., editor, *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 556–572. Springer Berlin Heidelberg, 2006.

Lobjois L. and Lemaître M. Branch and bound algorithm selection by performance prediction. In *Proceedings of the national conference on artificial intelligence*, pages 353–358. John Wiley & sons, 1998.

Locatelli M. and Wood G. Objective function features providing barriers to rapid global optimization. *Journal of Global Optimization*, 31(4):549–565, 2005.

López-Ibánez M., Dubois-Lacoste J., Stützle T., and Birattari M. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université Libre de Bruxelles, 2011. TR/IRIDIA/2011-004.

Lourenço H. R., Martin O., Stützle T., Glover F., and Kochenberger G. Iterated local search. *Handbook of Metaheuristics*, pages 321–353, 2002.

Lova A., Tormos P., Cervantes M., and Barber F. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. *International Journal of Production Economics*, 117(2): 302–316, 2009.

Maenhout B. *Exact and Meta-heuristic Algorithms for Nurse Shift Scheduling Problems*. PhD thesis, Universiteit Gent, 2007.

Mahajan Y. S., Fu Z., and Malik S. Zchaff2004: An efficient sat solver. In *Theory and Applications of Satisfiability Testing*, pages 360–375. Springer, 2005.

Maron O. and Moore A. W. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

Mastor A. A. An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science*, 16(11):728–746, 1970.

Messelis T., Haspeslagh S., Bilgin B., De Causmaecker P., and Vanden Berghe G. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux conference on Artificial Intelligence.*, pages 177–183, 2009.

Minton S. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1-2):7–43, 1996.

Misir M. *Intelligent hyper-heuristics: a tool for solving generic optimisation problems*. PhD thesis, Katholieke Universiteit Leuven, 2012.

Mitchell T. *Machine Learning*. McGraw Hill, 1997.

Musliu N. and Schwengerer M. Algorithm selection for the graph coloring problem. In *Learning and Intelligent Optimization (LION 7), Catania, Italy, Jan 7-11, 2013, to appear*. Springer, 2013.

Nonobe K. INRC2010: An approach using a general constraint optimization solver. https://www.kuleuven-kulak.be/~u0041139/nrpcompetition/abstracts/s2.pdf, 2010.

Nonobe K. and Ibaraki T. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In Ribeiro C. and Celso P., editors, *Essays and Surveys in Metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.

Nudelman E., Leyton-Brown K., Devkar A., Shoham Y., and Hoos H. Understanding random SAT: Beyond the clauses-to-variables ratio. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 438–452, 2004.

Osogami T. and Imai H. Classification of various neighborhood operations for the nurse scheduling problem. *Algorithms and Computation*, pages 72–83, 2000.

Ou J. Edge cuts leaving components of order at least m. *Discrete mathematics*, 305(1):365–371, 2005.

Özcan E., Bilgin B., and Korkmaz E. E. A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.*, 12(1):3–23, Jan. 2008.

O'Mahony E., Hebrard E., Holland A., Nugent C., and O'Sullivan B. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

Pascoe T. L. Allocation of resources - CPM. *Revue Francaise Recherche Operationelle*, 38:31–38, 1966.

Patterson J. H. Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, 23(1):95–123, 1976.

Pfahringer B., Bensusan H., and Giraud-Carrier C. Tell me who can learn you and i can tell you who you are: Landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, pages 743–750, 2000.

Preuss M. and Bartz-Beielstein T. Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, pages 91–119. Springer, 2007.

Pulina L. and Tacchella A. A multi-engine solver for quantified boolean formulas. In *Principles and Practice of Constraint Programming–CP 2007*, pages 574–589. Springer, 2007.

Pulina L. and Tacchella A. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.

Ramakrishnan N., Rice J. R., and Houstis E. N. GAUSS: an online algorithm selection system for numerical quadrature. *Advances in Engineering Software*, 33(1):27–36, 2002.

Reeves C. R. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.

Rice J. R. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Ridge E. and Kudenko D. An analysis of problem difficulty for a class of optimisation heuristics. In *Evolutionary Computation in Combinatorial Optimization*, pages 198–209. Springer, 2007.

Rosé H., Ebeling W., and Asselmeyer T. The density of states—a measure of the difficulty of optimisation problems. In *Parallel Problem Solving from Nature—PPSN IV*, pages 208–217. Springer, 1996.

Sassano A. On the facial structure of the set covering polytope. *Mathematical Programming*, 44(1-3):181–202, 1989.

Schiavinotto T. and Stützle T. A review of metrics on permutations for search landscape analysis. *Computers & operations research*, 34(10):3143–3153, 2007.

Selman B., Levesque H., and Mitchell D. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

Selman B., Mitchell D., and Levesque H. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

Smet P., Bilgin B., De Causmaecker P., and Vanden Berghe G. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, pages 1–24, 2012.

Smet P., Brucker P., De Causmaecker P., and Vanden Berghe G. Polynomially solvable formulations for a class of nurse rostering problems. In *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling, Practice and Theory of Automated Timetabling (PATAT2014), York, United Kingdom, 26-29 August 2014*, 2014.

Smith L. D., Wiggins A., and Bird D. Post-implementation experience with computer-assisted nurse scheduling in a large hospital. *Information Systems and Operational Research*, 17(4):309 – 321, 1979.

Smith-Miles K. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25, 2009.

Smith-Miles K. and Lopes L. Generalising algorithm performance in instance space: A timetabling case study. In Coello Coello C. A., editor, *LION*, volume 6683 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2011.

Smith-Miles K. and Lopes L. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

Smith-Miles K. and Tan T. T. Measuring algorithm footprints in instance space. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 3446–3453, 2012.

Smith-Miles K. and van Hemert J. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011.

Smith-Miles K., van Hemert J., and Lim X. Y. Understanding tsp difficulty by learning from evolved instances. In *Learning and intelligent optimization*, pages 266–280. Springer, 2010.

Sprecher A., Hartmann S., and Drexl A. An exact algorithm for project scheduling with multiple modes. *Operations-Research-Spektrum*, 19(3):195–203, 1997.

Stützle T. and Fernandes S. New benchmark instances for the qap and the experimental analysis of algorithms. In *Evolutionary Computation in Combinatorial Optimization*, pages 199–209. Springer, 2004.

Talbot F. B. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28(10):1197–1210, 1982.

Van Peteghem V. *Multi-mode Resource-constrained Project Scheduling Problem: Solution Procedures and Extensions*. PhD thesis, Universiteit Gent, 2010.

Van Peteghem V. and Vanhoucke M. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *J. Heuristics*, 17(6):705–728, 2011.

Van Peteghem V. and Vanhoucke M. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, 235(1):62 – 72, 2014.

Vancroonenburg W., De Causmaecker P., and Vanden Berghe G. A study of decision support models for online patient-to-room assignment planning. *Annals of Operations Research*, pages 1–19, 2013.

Vassilevska V., Williams R., and Woo S. L. M. Confronting hardness using a hybrid approach. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1–10. ACM, 2006.

Vollmann T. E. and Buffa E. S. The facilities layout problem in perspective. *Management Science*, 12(10):B–450, 1966.

Weinberger E. D. Local properties of kauffman's nk model: A tunably rugged energy landscape. *Physical Review A*, 44(10):6399, 1991.

White D. R. and Harary F. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, 31(1):305–359, 2001.

Witten I. H. and Frank E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

Węglarz J., Józefowska J., Mika M., and Waligóra G. Project scheduling with finite or infinite number of activity processing modes - a survey. *European Journal of Operational Research*, 208(3):177 – 205, 2011.

Woeginger G. J. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

Xu L., Hutter F., Hoos H. H., and Leyton-Brown K. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32: 565–606, 2008.

Xu L., Hutter F., Hoos H. H., and Leyton-Brown K. Satzilla2009: an automatic algorithm portfolio for sat. *SAT*, 4:53–55, 2009.

Xu L., Hoos H. H., and Leyton-Brown K. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10)*, pages 210–216, 2010.

Xu L., Hutter F., Hoos H. H., and Leyton-Brown K. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Xu L., Hutter F., Hoos H. H., and Leyton-Brown K. Features for SAT. Technical report, University of British Columbia, 2012a. http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/.

Xu L., Hutter F., Hoos H. H., and Leyton-Brown K. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. In *Proceedings of SAT Challenge 2012 : Solver and Benchmark Descriptions*, pages 57–58. University of Helsinki, 2012b.

Zhang W. and Korf R. E. A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, 81(1):223–239, 1996.

Zhu G., Bard J. F., and Yu G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing*, 18(3):377–390, 2006.

List of publications

Articles in internationally reviewed academic journals

- Messelis T., De Causmaecker P. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511-528, 2014.

Academic books; as editor

- De Causmaecker P. (ed.), Maervoet J. (ed.), Messelis T. (ed.), Verbeeck K. (ed.), Vermeulen T. (ed.). *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011), Gent, Belgium, 3–4 November 2011*, 2011.

Papers at international scientific conferences and symposia, published in full in proceedings

- Messelis T., De Causmaecker P., Vanden Berghe G. Algorithm performance prediction for nurse rostering. *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2013), Gent, Belgium, 27–28 August 2013*, pages 21–38, 2013.
- Haspeslagh S., Messelis T., Vanden Berghe G., De Causmaecker P. An efficient translation scheme for representing nurse rostering problems as satisfiability problems. *ICAART 2013 - Proceedings of the 4th International Conference on Agents and Artificial Intelligence, Barcelona, Spain, 15–18 February 2013*, 2013.

- Messelis T., De Causmaecker P. An algorithm selection approach for nurse rostering. *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011), Gent, Belgium, 3−4 November 2011*, pages 160−166, 2011.
- Messelis T., Haspeslagh S., Bilgin G., De Causmaecker P., Vanden Berghe G. Towards prediction of algorithm performance in real world optimisation problems. *Proceedings of the 21st Benelux Conference on Artificial Intelligence (BNAIC 2009), Eindhoven, The Netherlands, 29−30 October 2009*, pages 177−183, 2009.

Meeting abstracts, presented at international scientific conferences and symposia

- Messelis T., De Causmaecker P. An algorithm portfolio for nurse rostering. *International Federation of Operational Research Societies Conference (IFORS 2011), Melbourne, Australia, 10−15 July 2011*, 2011.
- Messelis T., Haspeslagh S., Bilgin G., De Causmaecker P. Algorithm performance prediction in a real world environment. *14th Belgian - French - German Conference on Optimization (BFG'09), Leuven, Belgium, 14−18 September 2009*, 2009.
- Bilgin B., De Causmaecker P., Haspeslagh S., Messelis T., Vanden Berghe G. Hardness studies for nurse rostering problems. *Learning and Intelligent Optimisation (LION 3), Trento, Italy, 14−18 January 2009*, 2009.

Meeting abstracts, presented at local scientific conferences and symposia

- Messelis T., De Causmaecker P. Automatic algorithm selection for multi-mode resource-constrained project scheduling problems. *27th Annual Conference of the Belgian Operations Research Society (ORBEL 27), Kortrijk, Belgium, 7−8 February 2013*, 2013.
- Messelis T., De Causmaecker P. Towards an algorithm portfolio for nurse rostering. *25th Annual Conference of the Belgian Operations Research Society (ORBEL 25), Gent, Belgium, 10−11 February 2011*, 2011.
- Messelis T., Haspeslagh S., De Causmaecker P. On expressing nurse rostering constraints as propositional satisfiability problems. *24th Annual*

Conference of the Belgian Operations Research Society (ORBEL 24), Liege, Belgium, 28−29 January 2010, 2010.

- Messelis T., Haspeslagh S., Bilgin G., De Causmaecker P., Vanden Berghe G. Hardness studies for nurse rostering problems. *23rd Annual Conference of the Belgian Operations Research Society (ORBEL 23), Leuven, Belgium, 5−6 January 2009*, 2009.

- Messelis T., De Causmaecker P. Distributed breakout algorithm for the distributed propositional satisfiability problem. *22nd Annual Conference of the Belgian Operations Research Society (ORBEL 22), Brussel, Belgium, 16−18 January 2008*, 2008.

FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE - KU LEUVEN KULAK
CODeS research group, member of ITEC-iMinds-KU Leuven
Etienne Sabbelaan 53, B-8500 Kortrijk